Inteligencia artificial con el ZX SPECTRUM



R. Jones M. Fairhurst

PARANINFO S A

Inteligencia artificial con el ZX SPECTRUM

ROBIN JONES

MICHAEL FAIRHURST

Inteligencia artificial con el ZX SPECTRUM

1986



MADRID

Traducido por
LUIS CAMPOY GUILLEN

Título original inglés:
ARTIFICIAL INTELLIGENCE: ZX SPECTRUM
publicado por SHIVA PUBLISHING LIMITED
64 Welsh Row, Nantwich, Cheshire CW5 5ES, England

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 1-85014-026-X (Ed. inglesa)
ISBN: 84-283-1488-8 (Ed. española)
Depósito Legal: M-26483-1986



Magallanes, 25 - 28015 Madrid (09303/34/86)

ALCO, artes gráficas. Jaspe, 34 - 28026 - MADRID

Indice de materias

Introducción

Quizás deberíamos comenzar diciendo lo que *no* es este libro. No es un libro de cocina. Ciertamente, contiene una serie de listados de programas que Vd. puede introducir en su Spectrum y ejecutarlos para obtener unos interesantes ejemplos de la utilización del ordenador para reconocimiento de patrones, bases de conocimiento, interacción conversacional, etc. Sin embargo, los resultados no serán "profesionales" en el sentido de que vaya a disponer de un buen soporte para gráficos, de menús detallados, páginas de ayuda; ni siquiera encontrará mensajes particularmente de ayuda. Es una política deliberadamente seguida en el curso de este libro.

Nos gustaría saber que Vd. ha comprendido un tema o, por lo menos, una sección del tema, antes de apresurarse a utilizar el teclado. De este modo, los programas ilustrarán y apoyarán sus conocimientos; no los ignore. Luego, haga lo posible por ampliar y profundizar sobre los temas que aquí se exponen. De hecho, en la mayor parte de los capítulos hay proyectos que sugieren ideas para una investigación complementaria. La mayoría de las rutinas son bastante cortas (diez a doce sentencias) que podrá usted tratar en trozos fragmentados, probarlas de modo independiente y quizás modificarlas para adecuarlas a sus propias necesidades antes de proseguir. La desventaja de esta solución es que quizás encuentre alguna dificultad para saber donde se encuentra en un determinado momento o quizás desee referirse a una rutina que no puede localizar fácilmente. Por ello, hemos incluido una serie de apéndices que contienen los listados totales de los programas principales. También incluyen las modificaciones marginales mencionadas, aunque no implantadas, del texto. No se hacen comentarios detallados ya que, después de todo, eso es una función del propio texto.

Ahora expondremos lo que *sí*, es el libro. Confiamos en que el libro sea una introducción moderada sobre algunas de las técnicas de la inteligencia artificial (IA) que se han venido desarrollando desde los últimos treinta años aproximadamente. No es una exposición superficial: existen temas a los que se les ha dado más o menos peso, que podrían merecer una discusión y otros que se han omitido enteramente. Alegaremos en nuestra defensa que en todas nuestras exposiciones nos hemos ceñido a los aspectos IA que se pueden implantar con facilidad en el ZX Spectrum. Insistimos: "con facilidad". No es nuestro deseo que se enfrente con grandes cantidades de código de máquina impenetrable. Para una mejor utilización de este libro necesitará disponer de cierta experiencia en BASIC y tampoco vendría mal tener cierto conocimiento del código de máquina, aunque no es absolutamente imprescindible. No necesita ser un mago del ensamblador Z80 para poder efectuar un seguimiento de lo que está sucediendo. No obstante, si dispone de esos conocimientos, podrá tomar muchas de las rutinas que presentamos y convertirlas a código de máquina, agilizando en gran medida las respuestas del Spectrum.

Hay una puntualización a hacer antes de comenzar. Es fácil involucrarse en exceso con un programa en particular y contemplar entusiasmado su comportamiento aparentemente inteligente. Sugerimos que cuide su sentido de admiración sobre la potencia de las actividades intelectuales humanas al compararla con las pequeñas pretensiones de los programas aquí presentados o con cualquiera de sus descendientes más potentes y sofisticados.

Isaac Newton decía: "No sé lo que pensará el mundo de mí, pero respecto a mí, pienso que he sido como un muchacho que se divertía en la playa tratando de buscar un guijarro bonito o una concha más bella de lo ordinario, mientras que ante mí estaba todo un gran océano de verdad por descubrir".

Todos los científicos deberían recordar esto, con más razón quienes están inmersos en una ciencia tan inmadura y joven como es la IA.

Hasta aquí hemos hablado de nosotros en plural. Sin embargo, cada uno de nosotros escribió algún capítulo en solitario, por lo que es más natural utilizar la primera persona "yo" en lugar de "nosotros"; así lo haremos en adelante, por lo que cuando digamos "nosotros deberá significar "El lector y yo".

1
¿Qué es la inteligencia artificial?

Parece adecuado comenzar definiendo nuestros términos. ¿Por qué algo o alguien es (o no es) artificialmente inteligente?

El término "artificial" es fácilmente comprensible. Una cosa, un dispositivo, un artefacto construido por el hombre puede tener inteligencia "artificial". El ser humano disfruta del término real. Pero, *¿cómo* muestra su inteligencia una persona o una máquina? Bien, procediendo inteligentemente, podríamos contestar, pero a nadie conformará esta respuesta; es una definición de tipo circular. Por tanto, vamos a tener que discurrir sobre lo que caracteriza a un comportamiento inteligente. Podríamos comenzar pensando en algunos ejemplos. Conducir un coche a todos se nos antoja que pueda ser una actividad inteligente. Esa inteligencia se muestra cuando reducimos nuestra velocidad al contemplar a un niño jugando con una pelota. Podría ocurrir que la pelota rodara a la calle y el niño tratara de ir a por ella. Contemplar esta escena e imaginarse Vd. mismo en una situación 30 segundos después, distinta de la que Vd. está *realmente*, parece también una actividad peculiarmente humana. En distintas áreas, esos 30 segundos pueden convertirse fácilmente en 30 minutos o en 5 años. Nosotros mostramos el mismo tipo de habilidad con los juegos: "Si ahora me enroco, él no me puede comer con su alfil el peón de rey, y si le deja allí y no mueve su rey, puedo comérmele con mi torre en el próximo movimiento y moverla después". O también, en temas más serios: "Si amplío mi casa, se incrementará el valor de mi propiedad, pero al mismo tiempo que obtendré una revalorización, costará más dinero calentar la casa".

Por otro lado, podemos pensar en actividades que no nos parecen nada inteligentes. Abrir una botella, etiquetar botes de conserva (en forma correcta, por supuesto), introducir una recarga de tinta en la pluma o el bolígrafo, todas estas actividades parecen triviales del ser

humano que pueden realizarse sin forzar nuestros músculos mentales en grado importante. Bien, pues *todas* estas actividades las podría realizar un brazo robot bajo control de un programa con bastante aproximación a la velocidad y destreza con que lo efectúa un humano y tendríamos la "intelligentsia artificial" (término a veces desacreditado utilizado para describir a quienes se encuentran investigando sobre la inteligencia artificial) rodando por las calles.

Por supuesto, *es* posible realizar todo eso "conduciendo un robot de la mano", es decir, enseñándole a base de guiar sus movimientos una vez, de forma que recuerde simplemente la secuencia de operaciones exactas controladas por motor y que posteriormente las repitiera exactamente. Rápidamente se dispondría a abrir botellas que no estuvieran a su alcance o etiquetaría dos botellas con una sola etiqueta si es que estaban demasiado juntas. A esto difícilmente se puede considerar como comportamiento inteligente.

Sin embargo, los programas de ajedrez los tenemos desde hace 20 años, y no hace falta que le diga que puede Vd. adquirir uno bastante decente para incorporarlo en cualquier micro casero. Estos programas se ejecutan en máquinas que tengan una potencia y velocidad razonables y pondrán en un compromiso a más de un jugador. Pero sin duda, no hay ningún programa de ajedrez que haya logrado batir todavía a un gran maestro, pero habrá millones de jugadores no profesionales a los que vencerá; estoy seguro de que esas personas se sentirán ofendidas si por ello pone Vd. en duda *su* inteligencia.

Por tanto, ¿un programa de juego de ajedrez es o no es "inteligente"? En apariencia parece que sí, ya que efectúa movimientos razonables, prevee las trampas que le pone su opositor e incluso él mismo pone también alguna trampa. Visto desde dentro, sin embargo, podemos ver cómo se consigue esto por medio de un procedimiento que utiliza enteramente estructuras arborescentes y evalúa así las posiciones del tablero en forma numérica (vea capítulo 9). Por tanto, se diferencia sólo en su mayor complejidad, respecto a un programa para cálculo de pagos de amortizaciones, que ciertamente no se puede considerar que caiga dentro del campo de la inteligencia artificial. A comienzos de los años 50, Alan Turing, matemático inglés y científico de ordenadores ya propuso un posible test para el comportamiento inteligente. Propuso que una persona se enfrentara con dos terminales de ordenador con los que podría conversar a través de los teclados. Uno de los terminales estaría conectado a un ordenador que se programaría para mantener una conversación con él. El otro estaría conectado a un segundo terminal operado por un ser humano. Si la persona no podía establecer la diferencia en calidad entre las dos conversaciones que sostenía, el ordenador habría pasado la prueba de inteligencia de Turing.

Sin embargo, hay quien nos quiere hacer creer que precisamente, debido a que podemos describir un proceso formal para resolver su problema, tal como el juego de ajedrez, no puede ser inteligente. Este argumento parece confundir la inteligencia con la intuición. Después de todo, un chispazo instantáneo de inspiración es un hecho claramente intuitivo y no existe un conjunto de procesos que podamos describir entre las premisas y la conclusión. Ahora mismo Vd. está comprometido en una actividad intuitiva. El reconocimiento simple de las letras de esta página requiere una técnica que ciertamente Vd. no acertaría a describir. ¿Cómo puede ser que acierte a reconocer una "p", una "P" o una "*p*", si en cierto modo significan lo mismo? Luego tiene que conjuntar las letras para llegar a formar palabras y finalmente tomar el sentido que incorporan las frases. Este es el momento auténtico del esfuerzo. Las ideas que presento aquí son bastante abstractas, si bien no debe tener (espero) dificultades en seguirlas. Pero Vd. no tiene idea de *cómo* está ensamblando esas ideas o incluso qué *significa* realmente formar un concepto.

Quizás la inteligencia tenga algo que ver con el nivel de abstracción involucrado en una actividad. Después de todo, disciplinas como la filosofía y las matemáticas tratan casi exclusivamente con ideas abstractas y tendemos a pensar que sus practicantes son particularmente inteligentes. Pero, ¿los números no son algo abstracto? El concepto "3 naranjas" puede ser bastante concreto pero el "3" en sí mismo es incapaz de tener una interpretación correcta. ¡Los ordenadores son especialmente buenos para el tratamiento numérico!

En términos de definición no ha habido más avances. De hecho podríamos argumentar que la frase "inteligencia artificial" se escogió de forma poco cuidadosa, dado que el concepto de "inteligencia" es algo abstracto. Pero lo tenemos adherido a nosotros casi del mismo modo que el teclado QWERTY que ya nadie (o casi nadie) cuestiona.

Por tanto, me gustaría proponer una definición que más o menos ignora del todo la palabra "inteligencia". Yo diría que quienes de algún modo están comprometidos con IA, intentan, de forma limitada, modelar algunos aspectos del comportamiento humano. Los juegos de sobremesa, la lectura de libros, abrir una botella, todas son actividades humanas de una forma que el cálculo de tablas de amortización no lo es, (se uqe voy a recibir varias cartas de queja por esta última afirmación; lo siento, pero Vds. saben a qué me estoy refiriendo).

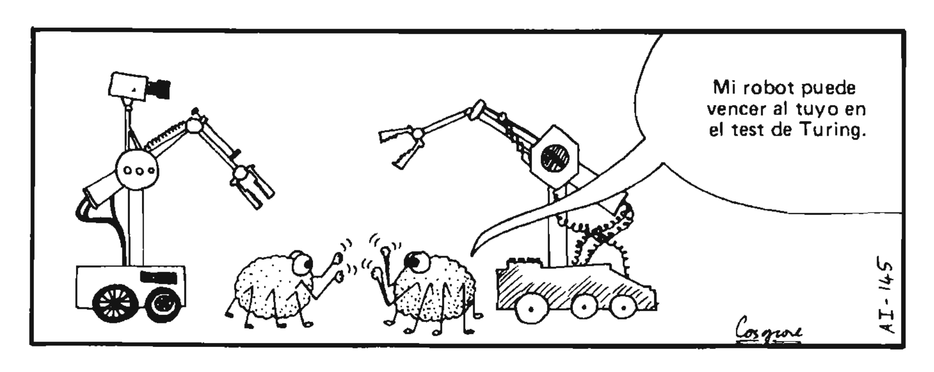
Esto es lo que para mí hace tan fascinante el área IA. El investigador puede desarrollar una técnica para resolver un problema y enfrentarlo luego a las posibilidades de realización del mismo problema por un ser humano o puede estudiar primero la técnica humana e intentar mode-

larla. En la última solución, el investigador comienza por aprender algo de un psicólogo. En la primera, el psicólogo puede aprender algo de él, ya que si el modelo desarrollado se produce para mostrar las características del tipo humano, puede utilizarse como hipótesis para la psicología humana del problema. En la práctica, puede existir mucho más diálogo del que aquí se sugiere entre el psicólogo y el hombre IA.

De nuevo, quizás estemos interesados en un modelo más "físico" del cerebro humano. En este caso hablaremos con un neurólogo en lugar de con el psicólogo, quien tiende a considerar el cerebro como una "caja negra" que sólo puede observar desde el exterior. Por tanto, es un área altamente interdisciplinaria.

Como ya ha sugerido, la modelación del comportamiento humano se presenta bajo diferentes perspectivas y premisas y de diferentes modos. Sin embargo, hay dos cosas que son comunes a todas las actividades humanas. Los humanos *aprenden* y luego *generalizan*. El aprendizaje es un proceso que nos es familiar incluso aunque el mecanismo por el que suceda se presente oscuro. Intentamos solucionar algo, falla, adoptamos una segunda solución, y funciona. Hemos *aprendido* una técnica. Por supuesto, alguien nos podría haber indicado que nuestra primera solución no funcionaría, en cuyo caso hubiéramos sido *enseñados*. Por tanto, nos interesaremos en examinar las formas en que un programa puede aprender por él mismo, así como las formas en que podemos enseñar a uno.

La generalización no es un término familiar, pero *es* una actividad común. Comenzamos con ejemplos sencillos de un concepto y luego hacemos hipótesis de naturaleza general. Esto se observa mucho en los niños. Indíquele a un niño lo que es un coche (rojo en este caso) y apuntará a un toldo rojo diciendo "coche". Se le ha indicado (implícitamente) que coche es rojo y el generaliza esto con la hipótesis de que



todo lo que sea rojo es un coche. Con mucha sensatez luego prueba su hipótesis gritando "coche" cada vez que ve algo rojo, si no se le indica, por supuesto, que está equivocado. El test ha fallado, la hipótesis es falsa. Con frecuencia tales hipótesis son ciertas o casi ciertas y el niño aprende mucho y muy deprisa. Por ejemplo, si se le enseña un Ford Fiesta y se le dice que es un coche puede formar la hipótesis de que "todas las cosas con cuatro ruedas son coches" que no está lejos de la verdad, necesitando sólo una pequeña revisión para que distinga los camiones, los coches de caballos, etc.

Examinaremos en breve las formas en que se pueden programar los ordenadores para imitar las características humanas, al menos de forma limitada.

2
Introducción al reconocimiento de patrones

El término "reconocimiento de patrones" es en sí mismo explicativo ¿o no lo es? Si se le pregunta qué significa para Vd., probablemente dirá algo como: "el reconocimiento de patrones describe lo que Vd. hace cuando reconoce un patrón, es decir, cuando examina algo, como un cuadro, y sabe qué es lo que está examinando". Bien, esto es válido en cierta medida, pero una definición tan simple oculta una multitud de problemas prácticos que nosotros, como seres humanos, con frecuencia damos por supuesto. ¿Se ha detenido a pensar cuántas actividades diarias implican este tipo de proceso que acabamos de describir?

El simple acto de leer estas palabras, requiere la posibilidad de reconocimiento de patrones. En realidad, no es tan simple como eso, ya que este "reconocimiento" sucede a distintos niveles. A nivel más bajo, para leer este libro Vd. debe ser capaz de reconocer que ciertos patrones de tinta y papel forman las *letras*, que son unos símbolos que, en términos de su experiencia anterior, son significativos para Vd. Estas letras generalmente significan poco por sí mismas, pero agrúpelas para formar palabras y frases y tendrá disponible gran cantidad de información. Sin embargo, para extraer esta información, Vd. debe conocer (reconocer) que una "S" es distinta de una "T", por supuesto, pero también que la secuencia de las letras L-A-S está permitida, mientras que la secuencia L-S-A no es una palabra válida.

Sin pensamos en las frases debemos aprender pronto que no todas las disposiciones de orden de palabras conforman frases válidas. Intente decir "¿Hora qué decirme puede es?" a un policía, por ejemplo, y probablemente le pida que se someta a la prueba del alcohol y sople por ese pequeño dispositivo o quizás le prepare una confortable cama para que pase Vd. la noche. En otras palabras, la estructura gramatical es importante y hay que añadirla ciertas reglas gramaticales (¿patrones?). Esta idea la desarrollaremos con más detalle en el capítulo 7.

Finalmente, antes de dejar nuestro ejemplo, es importante darse cuenta de que el concepto de reconocimiento se aplica incluso a un nivel más alto y que una palabra de una frase debe transmitirle el reconocimiento de una idea o concepto en particular o la ejecución de una función gramatical reconocible si su propósito es el de transmitir información tal y como se pretendía. En otras palabras, podemos escribir una frase gramaticalmente perfecta como:

"El mirlo rojo habló dentro de la felicidad fría" que no significa alsolutamente nada. ¿O no es así?

El otro punto muy importante a considerar en el reconocimiento de patrones que nosotros deberíamos seguir es que el término define una multitud increíble de actividades que, aunque todas ellas se encuentran involucradas con el concepto de reconocimiento, tienen que ver con diferentes aspectos del comportamiento. Nosotros somos capaces de reconocer las caras (la mayoría con dos ojos, dos orejas, una nariz, una boca y con frecuencia con pelo), y sin embargo, ¿por qué son tan diferentes? Podemos oir e interpretar sonidos, no sólo de una conversación, sino también los de una olla hirviendo, el tema de una sinfonía de Mozart y muchos más. Reconocemos los síntomas de un resfriado o una gripe y observamos la diferencia entre un Mercedes y un Seat 133. Podemos identificar a distintos locutores e incluso detectar en ellos esas pequeñas inflexiones en la voz que denotan angustia, sorpresa, miedo y otras sensaciones. Por si no fuera suficiente, piense en el proceso menos definible aún que seguimos en situaciones donde reconocemos el *estilo* de un compositor o las características de la trama de una película tipo Spaghetti Western.

Por tanto, queda claro que, según hemos visto, el reconocimiento de patrones es tan importante para el ser humano que es difícil pensar cómo podríamos existir en el mundo real sin esa posibilidad. Sin embargo, esta discusión es particularmente relevante en el presente contexto debido a dos razones.

La primera, que la discusión anteriormente expuesta no deja lugar a dudas de que el reconocimiento de patrones es un problema que requiere una inteligencia considerable y, siguiendo las ideas del capítulo 1, a cualquier máquina que incorpore una capacidad semejante en alguna o todas esas áreas se la puede considerar, con cierto grado de razón que se comporta "inteligentemente". En segundo lugar, la gran diversidad de tareas del reconocimiento de patrones sugiere que cuando desarrollemos las técnicas con las que abordar los problemas de reconocimiento de patrones que son en lo posible independientes del problema, vamos a encontrarnos además muy ocupados.

APLICACIONES DEL RECONOCIMIENTO DE PATRONES

Quizás esté pensando que todo esto es un trozo de arenque. El reconocimiento de patrones es muy interesante pero ello no significa que necesariamente haya de existir un punto en que se deba intentar programar un ordenador para llevar a cabo las mismas tareas, a menos que, por supuesto disfrute de la gimnasia intelectual. Por lo tanto, quizás es una buena idea que antes de adentrarnos en el reconocimiento de patrones de forma ligeramente más detallada, consideremos algunas aplicaciones del mundo real para los sistemas que ejecutan un reconocimiento de patrones.

El problema es que realmente existen tantas posibles aplicaciones que aparecen obvias con tan relativamente poca reflexión, que es difícil saber qué ejemplos se deberían escoger. He aquí simplemente algunas de las muchas aplicaciones posibles, para ilustrar algunas áreas de interés:

Procesos de fabricación

Bien, todos hemos oído de los coches construidos por las manos de los robots.

A pesar de que, según el contexto de la discusión del capítulo 1, muchos de los robots utilizados actualmente en fabricación no son realmente "inteligentes", ya que simplemente realizan una secuencia repetitiva de operaciones, existe un gran interés en la utilización de robots basados en sensores que son capaces de trabajar y captar el ambiente en el que trabajan. Un área típica de aplicación, existen muchas, es la inspección automática para detección de defectos en los productos y mejorar así el control de calidad. El **reconocimiento de patrones** involucrado aquí trataría de "reconocer" un componente (en una línea de montaje, por ejemplo) de calidad aceptable y distinguirle de los que se consideran defectuosos o inaceptables por alguna razón. Así mismo, se podrían utilizar en las fábricas robots que incorporaran un dispositivo sensor visual para localizar, identificar y acceder a un componente para efectuar un ensamblaje automático, luego transferirlo a una fase subsiguiente del proceso de fabricación, etc.

Reconocimiento de voz

Un gran apartado del reconocimiento de patrones está relacionado con el problema del reconocimiento de voz. El problema aquí es el de tratar de diseñar un sistema que pueda responder a una información ha-

blada, en lugar de a señales de control eléctricas o, como es el caso normal con un ordenador, señales obtenidas desde un teclado.

Al margen de que la interacción hombre-máquina sería más "amistosa", el reconocimiento automático de voz facilitaría mucho la vida, digamos por ejemplo, para los operadores de máquina que tienen ocupadas las manos; o podría facilitar un medio, potencialmente muy potente, de identificación individual a efectos de seguridad. El reconocimiento de voz abriría también un abanico de oportunidades para los minusválidos, ya que encontrarían en la comunicación oral mayor facilidad que en la operación manual de interruptores o del teclado.

Además, el reconocimiento de voz es un tema de un interés tan extendido que he dedicado un capítulo entero (capítulo 4) a la discusión de los principios más importantes.

Exploraciones Médicas

Los patrones en datos clínicos que describen un caso "normal" o "anormal" son muy comunes en la diagnosis médica o en el control de pacientes. El reconocimiento de patrones puede utilizarse como ayuda en el proceso masivo de imágenes de rayos X o para monitorizar un electrocardiograma que muestra la actividad eléctrica del corazón. En estos casos, en los que se hace absolutamente crítica una interpretación correcta, puede ser apropiado limitar el grado de la automatización de la detección de un aspecto sospechoso, o sólo potencialmente anormal, permitiendo que sea el doctor quien realice un examen final.

Proceso de Datos

Existen muchos procesos en el mundo de los negocios y el comercial que requieren la lectura de documentos o información escrita de distintos tipos. El reconocimiento de patrones podría aportar la base para disponer de una herramienta muy potente para aplicaciones tales como la lectura directa de facturas o pedidos, o la clasificación automática de cheques. Una lectura automática del código postal haría mucho más eficiente el tratamiento del correo, mientras que muchos procesos de gestión se beneficiarían de la posibilidad de entrada de datos directa por medio de la lectura automática de caracteres escritos a mano.

Ambientes Nocivos

Existen muchas situaciones en las que un ambiente de trabajo es tan peligroso que los operadores humanos no pueden funcionar adecuadamente dentro de él. En tales situaciones, o allí donde un operador deba trabajar por control remoto, el disponer de la posibilidad de reconocimiento de patrones sería algo muy positivo. Por ejemplo, en una planta química se podría detectar automáticamente la acumulación de una masa de gases explosivos o avisar de la presencia de una polución excesiva en alguna fase del proceso.

Hasta aquí y con la exposición anterior, sólo hemos arañado en la superficie; seguramente Vd. podría ampliar la lista expuesta, pero confío en que estará de acuerdo en que el reconocimiento de patrones es un tema importante y un sugestivo objeto de estudio.

SOLUCIONES PARA EL RECONOCIMIENTO DE PATRONES

A pesar de que en los capítulos 2 a 6 examinaremos el reconocimiento de patrones desde un punto de vista más práctico, es importante, en principio, disponer de un conocimiento más general de algunos de los conceptos subyacentes y los principios relevantes. Esto quizás se consiga mejor por medio de ejemplos; por tanto, consideraremos tres ejemplos (todos ellos con patrones *visuales* que representan caracteres alfanuméricos) que nos conducirán a las ideas principales con las que nos volveremos a encontrar más tarde para tratarlas de forma más rigurosa.

Ejemplo 1: Sistemas OCR (Reconocimiento óptico de caracteres)

Este tipo de sistema queda mejor ilustrado si consideramos el formato usual de un cheque bancario. Como muestra la figura 2.1 el cheque contiene en la parte inferior una serie de cifras que proporcionan información sobre el código de clasificación del banco, el número de cheque y el número de cuenta del cliente. Estos números impresos se utilizan para automatizar algunos de los procesos del tratamiento de los cheques y por tanto se necesita que sean legibles por una máquina. Así pues, ¿cómo solucionamos el problema de diseñar una máquina que lea estos datos?

Una solución sencilla que puede ser muy efectiva consiste en constuir una *máscara* para cada uno de los caracteres que se puedan dar (en la figura 2.2 se muestran algunos ejemplos). Para identificar un carácter se

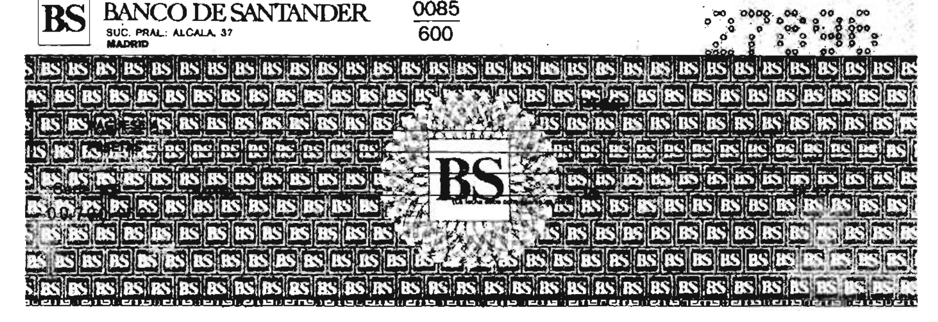


Fig. 2.1.—Formato usual de un cheque bancario.

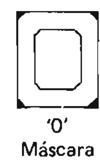
le compara simplemente con cada máscara hasta que se encuentre su semejante, que indicará su reconocimiento.

Es fácil ver que el patrón de prueba mostrado sería reconocido como un "cuatro", ya que coincide con la máscara correspondiente.

Vd. no necesita una perspicacia extraordinaria para ver que esta sencilla solución sólo es satisfactoria debido a que hemos aplicado algunas restricciones bastante severas al problema. Específicamente:

1. Los caracteres están perfectamente bien definidos y se han impreso utilizando un formato especial y generalmente aceptado al que están sujetos todos los usuarios potenciales del sistema.
2. Los caracteres utilizados se han seleccionado cuidadosamente (siendo el número de caracteres diferentes disponibles restringido) de forma que sean distinguibles rápidamente. Por esto todos ellos parecen ligeramente singulares, como si fueran de la era espacial.
3. Lo que puede no ser tan obvio pero igualmente importante es que las condiciones bajo las que se intenta el reconocimiento son también una gran







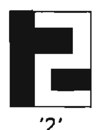








Fig. 2.2.—Ejemplos de máscaras de caracteres.

ayuda. Por ejemplo, sabemos siempre donde buscar los números en el cheque, todos ellos son del mismo tamaño, los cheques normalmente están bastante limpios, etc.

Parece pues que nuestra primera excursión al reconocimiento automático de patrones quizás no es tan impresionante como hubiéramos podido esperar. Además, se podría decir que el proceso que hemos examinado no es inteligente en absoluto, y yo probablemente estaría de acuerdo. Necesitamos ampliar un poco el área de problema para poder ver con más claridad los principios implicados.

Seguramente se habrá percatado de que en la tarea específica que hemos estado describiendo es bastante posible (y además común) obtener los datos que se van a clasificar no por la imagen óptica, sino de forma alternativa por medio de una tinta magnética especial. En este caso, por supuesto, en sentido estricto, no estaríamos tratando del reconocimiento de caracteres ópticos pero es claro que los principios serían exactamente los mismos.

Ejemplo 2: Reconocimiento de caracteres de imprenta

Consideremos ahora el problema de diseñar un sistema que necesitamos, digamos, para reconocer/distinguir un conjunto de caracteres alfabéticos impresos con una máquina de escribir, asumiendo que todos nuestros datos corresponden a ejemplos de letras del alfabeto.

A primera vista podríamos pensar que, al igual que en el ejemplo anterior, hemos restringido el problema lo suficiente como para poder aplicar una técnica de búsqueda de coincidencia (matching) similar a la utilizada en el caso OCR. Sin embargo, después de una pequeña reflexión, veremos que no es probable que en este caso sea satisfactorio. La esencia del problema es el hecho de que dos máquinas de escribir no siempre producen idénticos caracteres de escritura, e incluso algunos pueden ser muy diferentes. Cualquiera que haya podido observar la variedad de bolas de impresión que existen para las máquinas de escribir eléctricas, sabrá a lo que me estoy refiriendo. Como veremos, en la práctica ésta no es la única dificultad, pero de todos modos continuaremos. Por lo tanto, es probable que sin almacenar las máscaras correspondientes a un gran número de patrones de ejemplo para comparación, el método de búsqueda de semejanza de máscara no sea efectivo y realmente necesitemos encontrar una solución mejor y más sutil.

Un esquema más general y potencialmente mucho más flexible que la técnica de búsqueda de coincidencia con máscaras es la que presenta-

mos por medio de un tipo de modelo o estructura conceptual de un procedimiento de reconocimiento de patrones generalizado del siguiente modo. (Observe que a este nivel no nos preocupa demasiado cómo *construir* una máquina a partir de sus componentes físicos, sino que estamos luchando para encontrar un *procedimiento* que funcione).

En este esquema lo único que debemos asumir es que el proceso de reconocimiento puede dividirse en una serie de etapas separadas de forma conveniente y comoquiera que no estamos interesados todavía en *cómo* se efectuaría la implantación, podemos hacer que estos sub-procesos sean llevados a cabo por "demonios" que corresponderían a unos mecanismos físicos todavía sin especificar. Para definir el proceso total del reconocimiento, necesitamos identificar las cuatro etapas distintas (y por tanto cuatro tipos diferentes de demonios) ilustradas en la figura 2.3. Podemos pues asumir el proceso de reconocimiento de la forma siguiente:

Etapa 1. La primera etapa de proceso, que en realidad es obvia y fácil de comprender, consiste en observar la presencia de un patrón a procesar y el paso de estos datos a la siguiente etapa. Asumimos que esto lo hace un *Demonio Imagen* y es fácil ver que en la vida real pueden

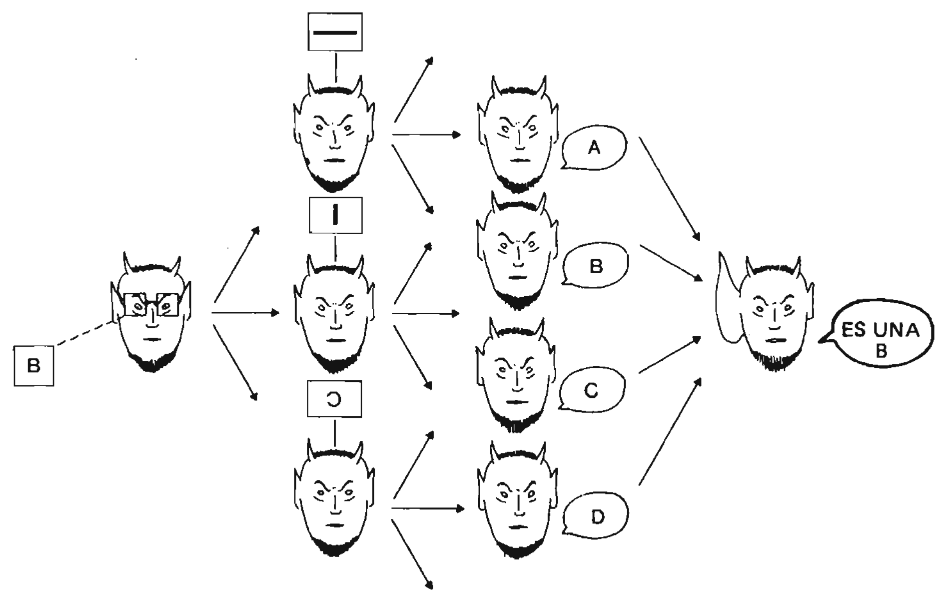


Fig. 2.3.—Pandemonium.

tener su répica en forma de cámara de TV o un dispositivo semejante que facilite una interfase entre una imagen y los datos que representa esa imagen.

Etapa 2. El Demonio Imagen pasa los datos de su imagen a un grupo de *Demonios de Rasgos*. Cada uno de estos Demonios de Rasgos tiene que investigar la imagen para detectar la presencia o ausencia de algún *rasgo* muy específico y bien definido. Podría ser, por ejemplo, una línea recta horizontal, una línea recta vertical, una curva, etc.

Etapa 3. Cada Demonio de Rasgos pone disponible, para la siguiente etapa de proceso, su propia información, de la que se hacen cargo los *Demonios Cognoscitivos*. Estos demonios son los responsables de buscar la ocurrencia de todo un conjunto de rasgos que se espera sucedan en cualquier clase específica de patrón. En este ejemplo lo que sucedería es que un Demonio Cognoscitivo buscaría todos los rasgos de una "A", otro buscaría los de la "B" y así sucesivamente hasta llegar a otro que se encargaría de buscar los de la "Z".

En este modelo asumiremos que un Demonio Cognoscitivo señaliza o indica la presencia de los rasgos que tiene asignados buscar por medio de un grito y cuanto más rasgos encontrara más fuerte gritaría.

Etapa 4. Un único *Demonio de Decisión* sería el responsable de la etapa final del proceso, teniendo a su cargo el nada envidiable trabajo de tener que escuchar los gritos de todos los Demonios Cognoscitivos para poder identificar cuál de ellos es el que grita más alto. Luego este demonio asigna el patrón de identificación (la imagen originalmente percibida por el Demonio Imagen) a la clase asociada con el Demonio Cognoscitivo que grita más alto.

No quedará sorprendido al conocer que este tipo de modelo fue bautizado originalmente como "Pandemonium". Como puede ver se concentra en la descripción de un procedimiento posible de reconocimiento de patrones en lugar de en una implantación en particular. Sin embargo, es importante darse cuenta de que al enfatizar que un patrón puede considerarse en términos de los rasgos de sus componentes en lugar de como una entidad única aislada, hemos seguido un camino más largo que la técnica original, que es claramente inadecuada para este tipo de aplicación.

Esto no quiere decir que el Pandemonium sea necesariamente la mejor solución ni siquiera que funcione en condiciones reales sin unas modificaciones complementarias. Como ejemplo ilustrativo, la tabla 2.1 muestra un ejemplo (pequeño) de tipos o clases posibles, junto a un conjunto de rasgos (más bien arbitrarios) que pueden escogerse. Para que vea Vd. algunos de los problemas que pueden aparecer en este

Tabla 2.1. Algunos tipos de caracteres y sus rasgos asociados

	F	L	O	P	R
Líneas verticales	1	1		1	1
Líneas horizontales	2	1		2	2
Líneas oblícuas					1
Angulos rectos	3	1		3	3
Angulos agudos					
Curvas continuas			1		
Curvas discontinuas				1	1

tipo de esquemas de reconocimiento, reflexione sobre las preguntas siguientes:

1. ¿Qué sucede cuando Vd. intenta incrementar el número de tipos distintos a reconocer?
2. Con patrones reales, ¿cómo es de sensible el sistema a la variabilidad de los rasgos?
3. ¿Qué dificultad existe para detectar los rasgos que Vd. haya escogido?
4. ¿Es posible siempre definir cada carácter posible de forma única?

Por supuesto, existen otras preguntas que Vd. deberá responder en situaciones particulares, pero parece que nos encontramos en el buen camino, ya que hemos encontrado un método, que si se aplica cuidadosamente, muestra cualidades como para pensar que pueda producir una técnica de reconocimiento generalizada.

Ahora sigamos con nuestro tercer ejemplo.

Ejemplo 3: Reconocimiento de alfanuméricos cualesquiera

Quizás éste sea el problema más general que deseemos solventar en el reconocimiento de caracteres. Supongamos que deseamos diseñar un sistema que reconozca cualquier carácter alfanumérico de toda la gama posible, las letras de la A a Z y los números de 0 al 9. Para evitar demasiada confusión, la letra O y el número 0 son el mismo carácter y la letra I y el número 1 tampoco debemos distinguirlos. Además, asumiremos que estamos tratando patrones en un ambiente operativo real (puede que para leer la dirección o al menos los códigos postales de unos sobres para el correo).

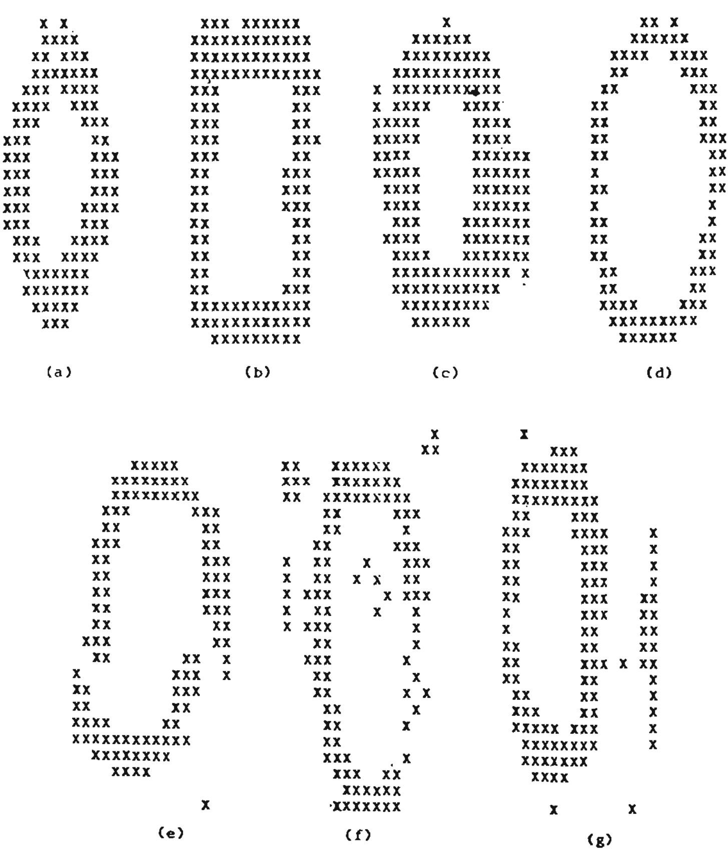
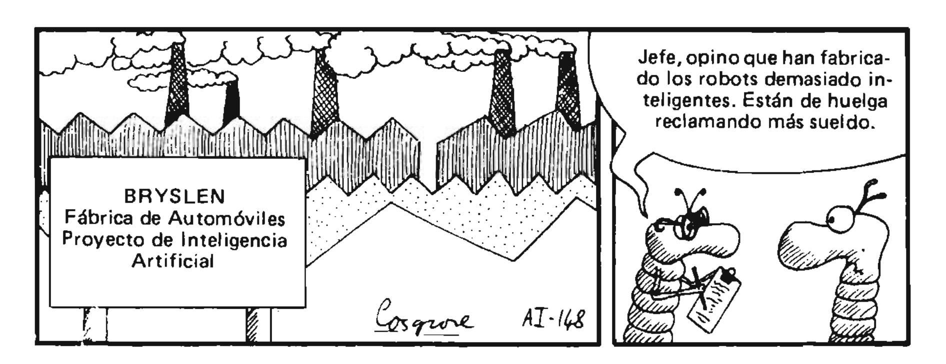


Fig. 2.4.—Ejemplos de la letra O impresa a máquina.

En este tipo de situación nos vamos a percatar de algunos pequeños problemas que hasta ahora habíamos dejado de lado. Una breve ojeada a la figura 2.4 servirá para ilustrar lo que quiero decir.

Aquí hemos tomados algunos patrones sencillos que corresponden a la letra "O" y está claro que incluso aunque supongamos que son caracteres escritos solamente a máquina, podemos esperar que existan algunas ligeras variaciones en la posible apariencia de los patrones a procesar (estos ejemplos se han extraido de datos reales de sobres de correo). Observe que estamos considerando unos patrones que se han convertido a una representación de "puntos en blanco y negro" para facilitar el proceso, pero de esto discutiremos en capítulos posteriores. Las figuras 2.4 (a) y (b) muestran la variación típica en la silueta del diseño del tipo de letra y en (c) y (d) también se muestra cómo incluso una imagen semejante puede aparecer en versiones más o menos gruesas. La figura 2.4 (e) muestra lo que puede suceder fácilmente al extraer los datos o al pasar la información a un sistema de ordenador; aquí, como ve, los caracteres quedan rotos y distorsionados. La figura 2.4 (f) ilustra el problema muy común del ruido en el patrón; en otras palabras, se han deslizado algunos puntos negros singulares (que surgen seguramente por las notas de suciedad en el papel, arrugas, etc.) mientras que otros puntos son blancos cuando debían ser negros (quizás debido a que la máquina de escribir tenía la cinta de carbón muy usada o el papel no absorbió debidamente la tinta). Finalmente, la figura 2.4 (g) muestra otro problema común; aquí la extracción del carácter para proceso no ha sido muy satisfactoria y ha arrastrado consigo parte del carácter adyacente.

Por lo tanto, verá que nuestras técnicas, aunque útiles para algunas situaciones específicas y razonablemente bien controladas, no son realmente apropiadas tal como están, cuando aparecen complicaciones. Por esta razón, necesitamos desarrollar unas técnicas que tengan más rigor.



Esto lo haremos en capítulos posteriores, pero al menos en este momento Vd. ya tiene una vaga idea de los principales problemas que hay que solucionar si queremos que el reconocimiento de patrones, esa actividad diaria que constantemente necesitamos, se lleve a los dominios de la práctica como implantación a través de los ordenadores.

3
Técnicas para el reconocimiento de patrones

Como vimos en el capítulo anterior, necesitamos desarrollar un poco más nuestras ideas sobre el reconocimiento automático de patrones, antes de abordar con éxito problemas del mundo real. En este capítulo intentaremos por lo tanto dirigirnos hacia otras técnicas generalmente aplicables y mejor definidas; pero para ello debemos comenzar por definir con algo más de precisión algunos términos y estar de acuerdo con algunas notaciones de expresión.

Comenzaremos con la idea de patrón. Aunque, como ya hemos visto, pensemos que conocemos intuitivamente lo que es un patrón, necesitamos precisar más para que sea de utilidad. Puesto que la idea del proceso de un patrón (por ejemplo, su reconocimiento) requiere que tengamos algo concreto que manejar, podemos comenzar observando que la generación de un patrón debe implicar una forma de medidas del ambiente (por ejemplo, obtención de datos en forma numérica). Esto puede comprender diferentes procedimientos ya que las medidas pueden corresponder a cosas como la intensidad de una luz en distintos puntos de una imagen visual (obtenida por ejemplo a través de una cámara de TV), la presencia o ausencia de los síntomas específicos en el diagnóstico de una dolencia, las frecuencias presentes en una onda sonora, etc. Sin embargo, lo más importante es que si nos concentramos en el hecho de que estamos midiendo una cantidad numérica que tiene alguna relación con el ambiente físico, podemos representar un patrón (cualquiera) como una cadena de medidas de los elementos de un patrón. Estos elementos son los que generalmente denominamos *rasgos* (features) de un patrón o *descriptores* de un patrón (ya que considerados juntos, "describen" el patrón) y ya podemos presentar así una notación o for-

ma de representación por la que se exprese un patrón (que llamaremos X) como su cadena de rasgos medidos, tales como:

$$X = (x_1, x_2, x_3, \ldots)$$

He aquí un ejemplo sencillo, al que volveremos más tarde, para ilustrar la idea. Examine la figura 3.1. Es la simple representación de la letra O en puntos en blanco y negro. Si ahora representamos esta imagen como una secuencia de puntos blancos y negros (de izquierda a derecha y de arriba a abajo) podríamos decir que nuestra paterna X se identifica como una cadena de 36 rasgos, donde en este caso cada rasgo es equivalente a un cuadrado de la parrilla mostrada y las medidas del rasgo tendrán un valor de 1 (para un punto negro) ó 0 (para un punto blanco). El patrón mostrado se podría escribir así:

$$X = (011110111111110011110011111111011110)$$

Observará que existen ciertas ventajas significativas al medir los rasgos de esta forma, en particular cuando deseamos continuar con el proceso de los patrones utilizando un ordenador.

Por supuesto, no es necesario restringirnos únicamente a la medida de rasgos tan simples ni a patrones tan obvios como las imágenes visuales del ejemplo que hemos tomado.

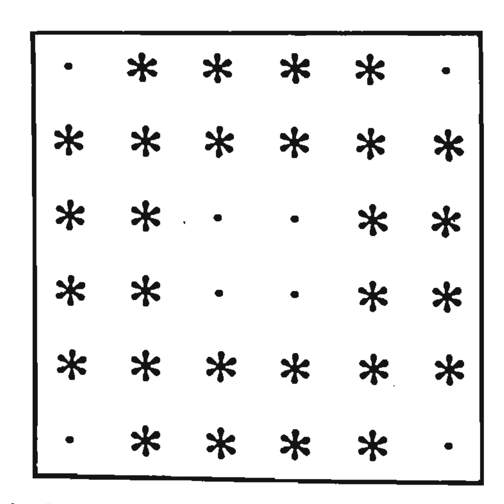


Fig. 3.1.—Representación de la letra O con puntos blancos y negros.

Tipos de patrones y su discriminación

Consideremos ahora otro ejemplo que no sólo aportará una ilustración complementaria del concepto de los rasgos o descriptores de los patrones sino que nos presentará también otra idea importante. El ejemplo que vamos a considerar trata de distinguir entre un hombre y una mujer. No hay que pensar demasiado para darse cuenta de que en principio no encerraría gran dificultad para un ser humano y que existen muchas formas, unas más obvias que otras, para poder establecer esa distinción. Sin embargo, nuestra intención es utilizar la idea de los rasgos medibles y yo supongo, que causemos un mínimo de perplejidad de acuerdo con nuestros patrones en la vida real. Por lo tanto, escogeremos dos parámetros medibles con los que trabajar; un buen punto de arranque podría ser tomar la altura y el peso como candidatos apropiados para nuestros rasgos seleccionados.

Si medimos estos rasgos para una serie de patrones ejemplo (personas) podríamos producir un gráfico como el que se muestra en la figura 3.2 que ilustra la relación altura/peso para una muestra típica de hombres y mujeres. Si representamos a lo largo de un eje la altura y sobre otro eje el peso y dibujamos los "hombres" como puntos + y los "mujeres" como puntos o, podemos ver que, al menos en este sencillo ejemplo, la posición de un patrón (la combinación de los rasgos de peso y altura medidos para un caso en particular) en este *espacio de rasgos* puede utilizarse para categorizar nuestra muestra. Esto es debido a que, según indica la figura 3.2 todos los puntos que definen la categoría hombres son distintos de los puntos de la categoría mujeres y, para probarlo, podemos trazar una línea que separe ambos grupos.

Ahora, por supuesto, si tomamos más muestras es probable que estos dos grupos puedan llegar a ser algo menos distintos y, además puede existir incluso cierto grado de solapamiento (todos hemos visto a las mujeres lanzadoras de disco), pero en gran medida es posible establecer que la distinción entre las dos categorías (tipos de patrones, si lo prefiere) se mantenga sustancialmente.

Ahora, si lo contemplamos aunque sea con una pequeña tendencia matemática, podemos ver que en este caso es muy fácil definir una regla de reconocimiento con la que decidir, para un ejemplo desconocido (representado por un par de medidas para peso y altura), si debiéramos clasificar una determinada muestra como hombre o como mujer. Aquí la regla es sencilla de ver, ya que el límite entre las dos clases o tipos es una línea recta matemáticamente muy conveniente. Este tipo de superficie de decisión o función discriminante, como se denomina frecuentemente, puede utilizarse, por lo tanto, para formular una regla de clasifi-

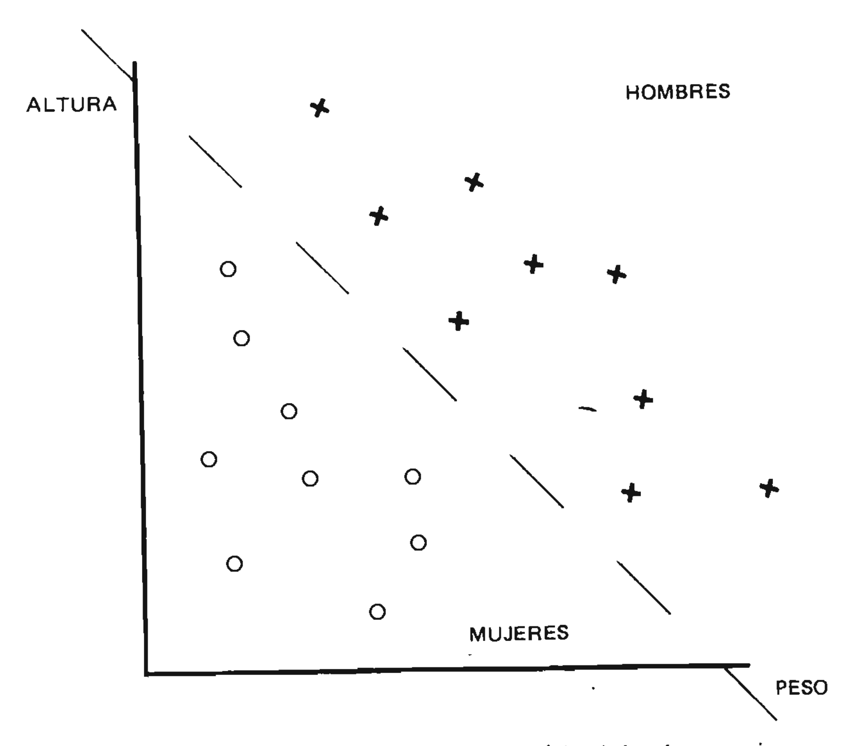


Fig. 3.2.—Relación peso/altura para una muestra típica de hombres y mujeres.

cación calculando simplemente a qué lado de la línea cae un punto en particular (en breve veremos un ejemplo de esto). Comprobará que en algunas situaciones más complejas una simple línea recta puede no ser suficiente y se requiere una forma diferente de *función discriminante*; pero el principio sería el mismo.

Este ejemplo también indica algunos otros aspectos importantes del reconocimiento de patrones. Por ejemplo, está claro que el gráfico dibujado es posible en principio, debido a que hemos realizado medidas específicas sobre los grupos de patrones que esperamos identificar. Aquí escogemos (de modo arbitrario) la altura y el peso y, a juzgar por nuestros resultados, estos parámetros parecen ser una elección razonable para esta tarea. La elección de rasgos puede ser crucial para el diseño satisfactorio de un sistema de reconocimiento de patrones y debe considerarse con sumo cuidado. Debe tener en cuenta lo siguiente:

1. ¿Hubiese sido satisfactorio haber elegido otros dos parámetros, como por ejemplo el número de dientes y la longitud del pelo, como los rasgos a utilizar para esta tarea?
2. ¿Serían igualmente apropiados los rasgos que hemos utilizado, si hubiéramos tratado de distinguir unos jugadores de una Liga de Baloncesto de unos jugadores de la Liga de Fútbol?

También se habrá fijado en que para tareas más complejas se necesitaría un mayor número de rasgos para poder realizar una decisión de reconocimiento. El problema entonces habría que representarlo en un espacio dimensional mayor (cualquier espacio con más de tres dimensiones se llama "hiperespacio"; sí, no se asombre, existe), y la función discriminante asumiría una forma matemática relativamente compleja, en lugar de la forma de línea recta fácilmente visualizable de nuestro ejemplo. Vd. podría ver sin gran dificultad la forma de ampliar esta idea a tres dimensiones.

Por supuesto, también verá que, en general, para determinar la línea divisoria entre los tipos (es decir, para determinar las características de la función discriminante), necesitamos algunos ejemplos con nombre y etiqueta de los tipos que debemos identificar.

Podemos ver, entonces que, en algunos casos, por ejemplo, en los casos sencillos de reconocimiento de dos tipos, una función matemática sencilla (aquí una ecuación de una línea) puede ser todo lo que se necesite para poder discriminar entre patrones que pertenecen a tipos diferentes. Tal y como precisamos anteriormente, podemos literalmente "ver" a qué clase pertenece un patrón trazando un diagrama donde se han dibujado los rasgos, dibujando la línea recta divisoria y encontrando en qué lado de la línea cae el patrón en cuestión.

Por supuesto que existe el problema de que, generalmente, no deseamos tener que dibujar monótonamente el cuadro cada vez que queremos clasificar un patrón nuevo y, por tanto, como alternativa, utilizaremos las propiedades matemáticas de la función discriminante. En la figura 3.3, se ilustra esto brevemente. Aquí, de nuevo, hemos dibujado dos grupos de patrones, cada uno de ellos representado por sólo dos rasgos, como antes. De nuevo hemos deducido una función discriminante, dibujando en los ejes de los rasgos una línea recta que divide los dos grupos de patrones. Sin embargo, podemos ser más precisos, ya que podríamos escribir una expresión para la relación general entre los descriptores x_1 y x_2 correspondientes a la ecuación de la función discriminante de la línea recta que hemos utilizado. Puesto que estamos tratando con una línea recta, esto es sencillo y para este caso, la ecuación de la línea sería:

$$2x_1 - x_2 - 2 = 0$$

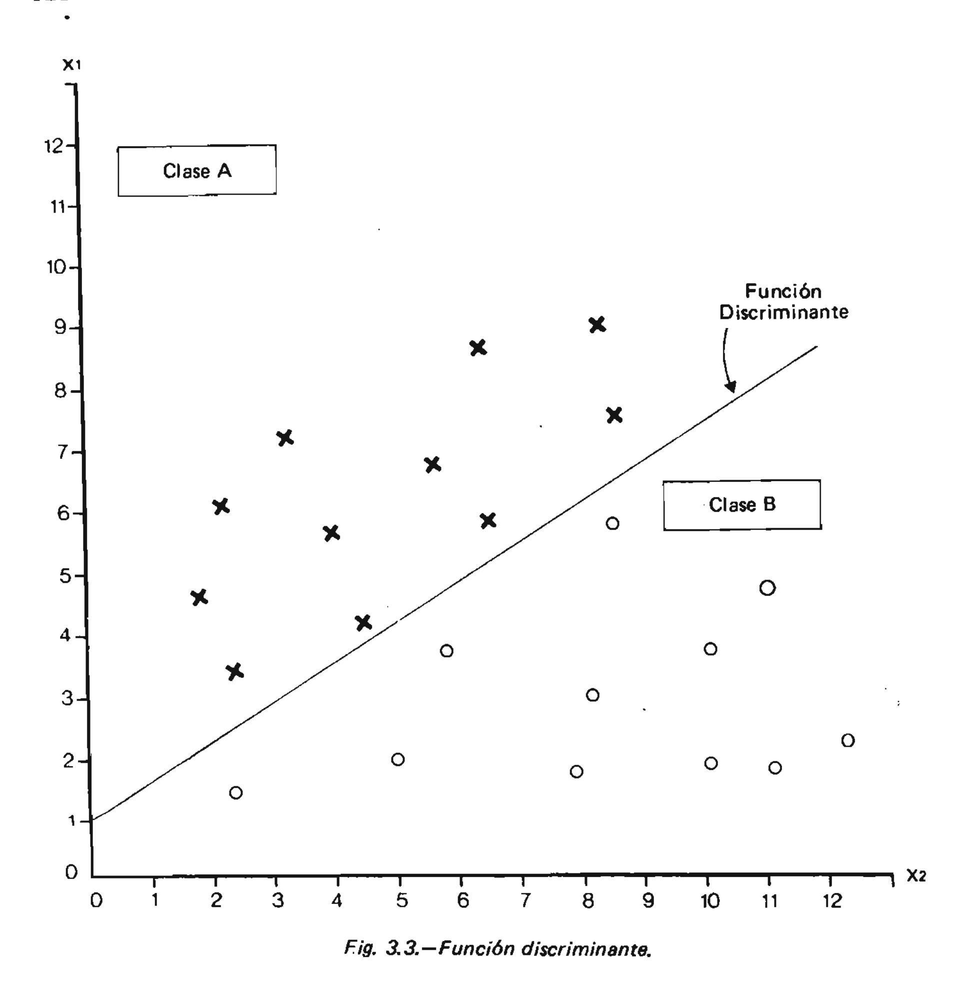


Fig. 3.3.—Función discriminante.

Esta ecuación representa la línea divisoria entre los dos tipos o clases. Dicho de otro modo, cualquier patrón que tenga medidas de descriptores que satisfagan exactamente a la ecuación de arriba ($x_1 = 5$ y $x_2 = 8$, por ejemplo), serían teóricamente inclasificables, ya que caerían exactamente sobre la línea.

Por otro lado, cualquier patrón con los valores x_1 y x_2 tales como:

$$2x_1 - x_2 - 2 > 0$$

caería por encima de la línea, tal como la hemos dibujado en la figura 3.3 y por tanto se supone que pertenece a la clase A, mientras que cualquier patrón con valores para x_1 y x_2 tales como:

$$2x_1 - x_2 - 2 < 0$$

caerían debajo de la línea del gráfico y por tanto se clasificarían como pertenecientes a la clase B.

Así pues, verá que nuestro *procedimiento de reconocimiento* podría describirse como sigue:

Etapa 1. Obtener los valores de los descriptores de patrones x_1 y x_2, para un patrón desconocido representado por (x_1, x_2).

Etapa 2. Sustituir estos valores en la ecuación de la función discriminante, dada por:

$$f(X) = 2x_1 - x_2 - 2$$

Etapa 3. Si $f(X) > 0$ asignar el patrón a la Clase A.

Etapa 4. Si $f(X) < 0$ asignar el patrón a la Clase B.

Vd. no necesita ser un gran matemático o un genio de la programación para llevar esto a la práctica, ¿no es así?

Por supuesto, este método funcionará para funciones discriminantes que puedan representarse mediante funciones no lineales, permitiendo así el reconocimiento de patrones que no caen tan claramente dentro de dos grupos como en el caso de nuestro ejemplo, pero estas funciones discriminantes son mucho más difíciles de deducir y manejar.

Si Vd. ha pensado cuidadosamente lo que hemos estado haciendo hasta aquí, se habrá percatado de que a menos que tengamos mucha suerte con los datos que tenemos que procesar, algunos de los puntos que dibujamos podrían caer muy cerca de la línea que representa la función discriminante. Para estos puntos tendríamos poca confianza en la decisión de clasificación, ya que la decisión sería marginal. Además, podríamos encontrarnos el caso de que algunos puntos cayeran a un lado en particular de la línea en lugar de en el otro simplemente porque hubiéramos medido los valores de los descriptores con algo menos de precisión de la que hubiéramos podido hacerlo. Entonces Vd. podrá pensar que es una buena idea intentar asegurarse que se tiene un grado razonable de confianza en la decisión que toma. ¿Cómo podría Vd. conseguir esto?

Antes de continuar, si examina la tabla 3.1 verá algunos datos de ejemplos complementarios que he incluido para que ejercite Vd. por su

Tabla 3.1.

		Número de patrón											
Clase A		A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}
	x_1	30	35	40	35	35	45	45	50	55	60	55	60
	x_2	25	75	60	45	35	40	55	65	70	60	50	65
Clase B		B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}
	x_1	30	40	45	55	65	55	65	70	80	70	85	80
	x_2	15	20	31	31	35	40	45	25	20	55	45	40

cuenta. Vea si puede Vd. deducir una función discriminante apropiada utilizando aproximadamente la mitad de los patrones y en qué afecta esto a la clasificación del resto.

Esquema alternativo de clasificación

Finalmente, en este capítulo, presentaremos otra solución para el reconocimiento de patrones que no depende tanto de la interpretación "visual", a pesar de que nuevamente su éxito depende de la suposición de que los miembros de un tipo tenderán a "agruparse" en base a sus medidas de rasgos, mientras que las agrupaciones para tipos diferentes tenderán a ser distintas. Esta técnica llamada de agrupación es muy atractiva ya que utiliza la idea de *distancia* entre los patrones, y la distancia es un concepto con el que estamos familiarizados intuitivamente.

Naturalmente, si vamos a utilizar el concepto de *distancia* como parámetro definido y medible en lugar de en un sentido puramente intuitivo, debemos acordar lo que significa el término. Aquí el problema es que en reconocimiento de patrones podemos definir la "distancia" de muchas maneras, aunque una medida utilizada comúnmente es la llamada distancia Euclídea (que en realidad es el mismo tipo de distancia que Vd. mide normalmente con una regla sobre un papel). Utilizaremos la palabra "distancia" para referirnos, a partir de ahora, a la distancia Euclídea y por tanto debemos definir cómo medir su valor en un contexto de reconocimiento de patrones.

Supongamos que consideramos dos únicos patrones, cada uno de ellos representado por dos únicos descriptores como anteriormente. Si etiquetamos el primer patrón como $P_1 = (x_{11}, x_{12})$ y el segundo como $P_2 = (x_{21}, x_{22})$, la distancia (d) entre ellos vendrá dada por:

$$d = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2}$$

Profundizando, Vd. puede demostrar que esto se corresponde con nuestro concepto "normal" de distancia, trazando unos pocos puntos en una hoja de papel gráfico o simplemente aplicando el Teorema de Pitágoras (suponiendo siempre, por supuesto, que Vd. lo recuerda).

También debe observar que podemos ampliar esta definición para tener en cuenta tantos rasgos o descriptores como necesitemos utilizar para cualquier tipo particular de patrón. Por ejemplo, si se han elegido tres descriptores, se calculará como:

$$d = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 + (x_{13} - x_{23})^2}$$

y así sucesivamente.

Entonces, ¿cómo utilizamos esta idea de distancia entre patrones como base para el reconocimiento de algunos patrones de identidad desconocida? Bien, podemos comenzar asumiendo que (como antes) tenemos disponibles algunos ejemplos etiquetados e identificados de los tipos de patrones que estamos tratando. Empleando la técnica de agrupación para reconocer un patrón desconocido, es relativamente sencillo calcular su distancia con todos los patrones de ejemplo que tenemos y asignarla al tipo en que se encuentra el patrón más cercano a ella, con arreglo a nuestro criterio de distancia elegido. Podríamos expresar el procedimiento en términos del flujograma de la figura 3.4 y Vd. debería estar capacitado para escribir por sí mismo un programa que realizara el procedimiento.

Antes de abandonar la idea de agrupación, es interesante destacar otra característica útil de esta solución y es que puede permitirnos considerar la posibilidad de clasificar patrones incluso si no disponemos de ningún ejemplo etiquetado de miembros típicos de una clase. En esta situación, nos podemos encontrar enfrentados a un conjunto de patrones de identidad desconocida que necesitamos dividir en categorías de acuerdo con sus propiedades comunes. Utilizando un procedimiento de agrupación podríamos examinar uno a uno cada patrón y asignarlo a una agrupación si su distancia desde cualquier agrupación actualmente disponible es menor que cualquier valor de entrada elegido arbitrariamente. Si un patrón se encuentra fuera de este límite de entrada enton-

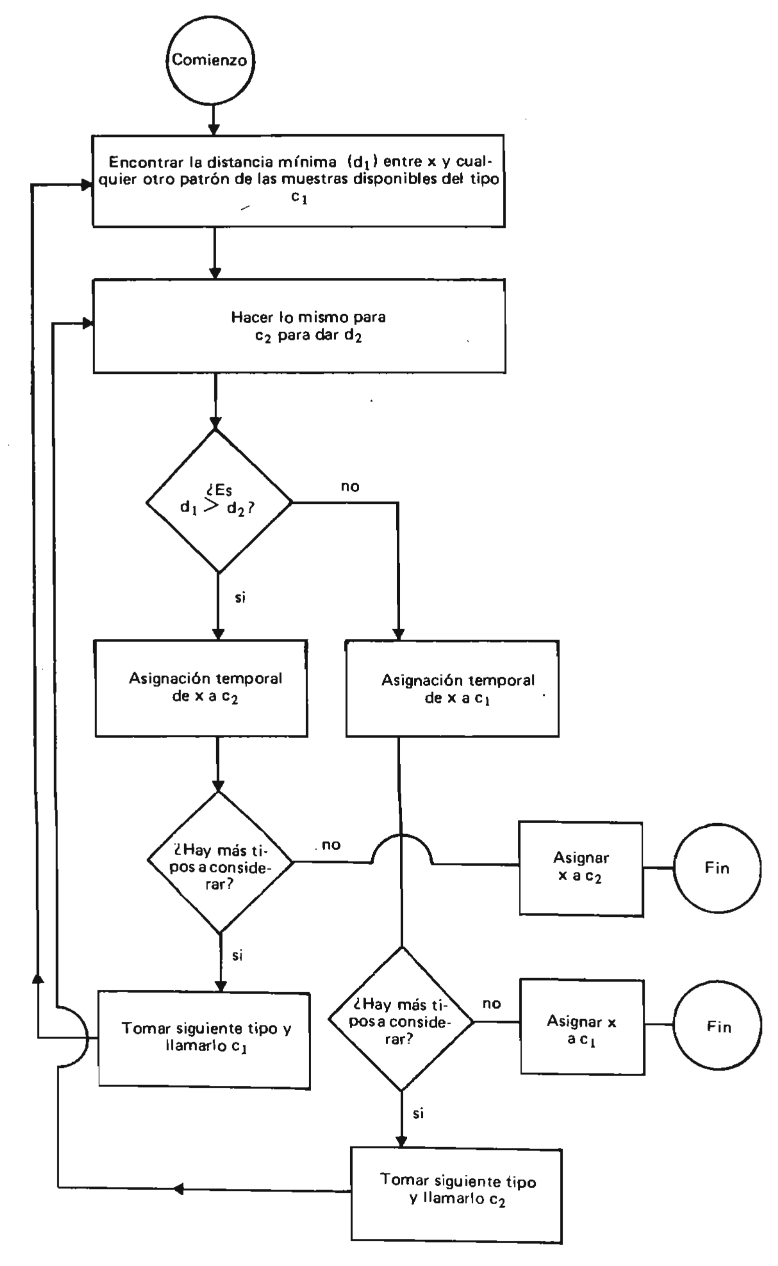


Fig. 3.4.—Flujograma para mostrar la técnica de agrupación.

ces se utiliza para formar una nueva agrupación; de esta forma podemos dividir nuestro conjunto de patrones en un conjunto de agrupaciones de tipos de patrones. Por supuesto, hemos obviado la pregunta de cómo escoger el límite de entrada para los miembros de un tipo, pero esta aplicación puede ser un ejemplo muy útil de este tipo de técnica.

En este capítulo hemos pasado de las ideas básicas y la teoría del reconocimiento de patrones a algunas técnicas sencillas que pueden aplicarse en una situación práctica. Aunque hemos utilizado unos ejemplos sencillos para ilustrar nuestros procedimientos, pienso que Vd. ya puede comenzar a ver cómo estas técnicas pueden extenderse a situaciones más complejas. Sin embargo, tenga cuidado, ya que el reconocimiento de patrones puede ser un proceso muy complejo y a menos que los problemas estén muy bien definidos, puede que le sucedan cosas extrañas. No se desanime, ya que precisamente este aspecto del reconocimiento de patrones es el que hace que en este área de trabajo exista mucha competencia y también recompensas y ello explica porqué tantos científicos están tratando todavía de investigar los problemas fundamentales de este tema además de explorar nuevas y excitantes aplicaciones de las técnicas ya establecidas.

Continuamos ahora con una tercera solución para el reconocimiento de patrones que proporcionará algo más de flexibilidad y presentará también una discusión más detallada sobre cómo tratar patrones en su ordenador. Para ampliar esto, por supuesto, debería intentar encontrar la distancia Euclídea más corta al capítulo 5...

4
Reconocimiento de voz

En nuestra discusión de introducción en el capítulo 2 sobre las ideas básicas del reconocimiento de patrones, yo mencioné específicamente el problema del reconocimiento de voz, haciendo hincapié en el hecho de que este área particular es de gran interés e importancia por sí misma. En este capítulo discutiremos algunos de los principales puntos de este área y, más específicamente, demostraremos cómo el reconocimiento de voz, aunque se encuentra ciertamente en el campo del reconocimiento de patrones requiere un proceso especializado.

El problema básico radica en el hecho de que, a diferencia del reconocimiento de patrones y las tareas que esto implica, tales como imágenes visuales de caracteres alfabéticos o incluso imágenes más complejas, como radiografías con rayos X o imágenes tipo TV en las que los rasgos y las características se pueden definir con relativa facilidad, por su propia naturaleza, la voz es algo difícil de definir de forma precisa tal y como hacíamos en los ejemplos anteriores. Existen dos motivos fundamentales donde radica la dificultad. En primer lugar, a diferencia de la imagen visual, que es algo que se puede considerar en términos bastante concretos en el sentido de que Vd. generalmente la puede dibujar en un papel y puede medir los parámetros apropiados que le permitan almacenarla de forma bastante natural dentro de un ordenador, la voz, como decíamos es un concepto más vago y más abstracto. En segundo lugar, la voz intrínsecamente varía con el tiempo. Es decir si Vd. intenta localizar la voz en un momento dado, entonces destruye su cualidad esencial por propia naturaleza. La voz sólo se define en términos de la variación en el tiempo y esto es más bien distinto de lo que hemos visto hasta ahora.

Inevitablemente, por tanto, para aplicar el tipo de técnicas que hemos discutido previamente a los datos que constituyen la voz necesita-

mos tratar nuestros datos en forma diferente de la adoptada hasta ahora. Desgraciadamente, en muchos aspectos se descuidan y se subestiman estas dificultades. La mayoría de los escritores de ciencia ficción se han percatado de que sin un ordenador que hable y entienda, su última historia de aventuras está en cierto modo incompleta; aunque, en general, no nos sentimos interesados por los pensamientos futuristas, la adopción ampliamente difundida de la idea se convierte gradualmente en una forma de pensar que sugiere que el reconocimiento de voz no es fundamentalmente más dificultoso que la mayoría del resto de tareas de ordenador. Además, estos problemas generales se combinan con frecuencia con el hecho de que en el uso normal del lenguaje natural, los seres humanos utilizamos unos giros en la conversación y unas construcciones altamente coloquiales que incluso para algún nativo de la lengua serían de difícil comprensión, y menos aún para una estúpida máquina.

Antes de seguir adelante, me gustaría apuntar que en este libro no va a ser posible explicar en detalle los aspectos técnicos del reconocimiento de voz, pero mostraremos cuáles son los problemas que comporta. También puede que Vd. se percate de que mucho de lo que aquí se expone está estrechamente relacionado con todo el mundo de la comprensión y entendimiento del lenguaje natural, que más tarde trataremos en este libro.

Algunos principios básicos del reconocimiento de voz

Al acercarnos al área del reconocimiento de voz es necesario clarificar algunos puntos para evitar confusiones y establecer una observación importante necesaria para diferenciar entre la idea de *reconocimiento* de voz como tal y *comprensión* o entendimiento de la voz.



El objetivo principal del reconocimiento de voz, al menos en lo que concierne a este capítulo, es el de generar una señal en respuesta a una expresión hablada directamente relacionada con el sonido expresado. Dicho de otra forma, el reconocimiento de voz solamente busca categorizar una entrada hablada dentro de un rango de tipos posibles. El objetivo de un sistema de comprensión de la voz iría mucho más lejos, en el sentido de que, para comprender una expresión hablada, se requiere no sólo que el sistema categorice un sonido, sino que responda también a él, de forma apropiada al contenido o significado de esa expresión.

Debemos poner cuidado con la utilización de la palabra "comprensión" ya que Vd. podría argumentar, con cierta justificación, que una máquina no puede comprender realmente nada (volvemos de nuevo a la vieja discusión filosófica), pero aquí hablamos solamente de comprensión en el sentido en que se le puede decir a un ordenador que comprenda un programa para sumar dos números. ¡Quizás sea mejor dejar esta engañosa cuestión antes de que nos quememos!

En cualquier caso pienso que Vd. probablemente se da cuenta de que el *reconocimiento* de voz es sólo una parte de lo que se podría llamar un sistema de comprensión de voz, pero una parte muy necesaria. Luego nuestro objetivo puede ser simplemente llegar a un punto en el que nuestro sistema reciba una entrada hablada y la traduzca a una forma escrita o impresa o que produzca algún otro tipo de representación de la expresión que sea adecuado para un proceso posterior (por ejemplo, para intentar extraer su "significado"). Si podemos llegar hasta aquí, podemos decir que hemos reconocido la voz.

PRODUCCION DE SEÑALES DE VOZ

La producción de la voz humana es un pequeño milagro de ingeniería mecánica. El aire reservado en un depósito apropiado (los pulmones) se expele a través de distintas cavidades tales como la boca, la nariz, etcétera, produciendo unas excitaciones y vibraciones que tienen por resultado la emisión de un sonido determinado. Estos sonidos se pueden hacer más interesantes si además se producen simultáneamente algunas oscilaciones en las cuerdas vocales. Este proceso es muy similar a la forma en que funciona la mayoría de los instrumentos musicales, ya que el sonido real emitido dependerá de las resonancias producidas en unas cavidades apropiadas. En el caso humano, por supuesto, estas cavidades pueden variar de forma por medio de unos movimientos hábiles de la boca, los labios, la lengua, etc., produciendo así una gama muy amplia de sonidos diferentes.

Es útil poder referirse a algunos patrones de sonidos específicos fundamentales que constituyen la base de nuestro lenguaje hablado; con arreglo a esto es posible identificar un grupo de sonidos específicos o *fonemas* a partir de los cuales se genera el lenguaje. He aquí algunos ejemplos de fonemas:

El sonido de una "a" pronunciado en la palabra "casa".

El sonido de una "c" pronunciado en la palabra "pico".

El sonido de una "c" pronunciado en la palabra "cielo".

El sonido de una "t" pronunciado en la palabra "pato".

y así sucesivamente.

Para complicar las cosas, nosotros generalmente etiquetamos los fonemas con símbolos especiales a efectos de identificación.

Quizás deba observar que los fonemas se pueden agrupar en categorías cuya disposición depende de la forma en que el grupo total de fonemas se produce físicamente.

Sonidos *fricativos* tales como la f surgen al expeler aire a través de un paso constreñido antes de excitar una resonancia en la cavidad, pero si se añade una vibración de las cuerdas vocales, el fonema se convierte en una V. La categoría de fonemas que produce el sonido más alarmante y potencialmente peligroso contine los sonidos *explosivos**. Estos sonidos se producen cerrando la boca, creando una cámara de aire a presión y luego soltando la presión con una vibración simultánea de las cuerdas vocales. Afortunadamente para las personas esto es más fácil de realizar que de describir. Un ejemplo de un *explosivo* es el fonema b.

CARACTERISTICAS DE LAS SEÑALES DE VOZ

Es bien sabido que cada fonema se produce por una única combinación de la actividad muscular del aparato vocal, pero ello no nos ayuda demasiado para poder diseñar un sistema de reconocimiento (aunque quienes son capaces de leer en los labios, intentan hacer esto exactamente). Lo que necesitamos en realidad es un medio de obtener *descriptores de patrones* para la voz, de forma semejante a la que adoptamos para patrones visuales. Para tener una idea de cómo solucionar esta cuestión, necesitamos penetrar con un poco más de profundidad en la naturaleza de la propia señal de la voz y hacernos algunas pregun-

* En fonética se acostumbran a denominar *bilabiales* la "b" y la "p". (N. del editor).

tas sobre el resultado de la actividad muscular que genera un sonido hablado.

Cada sonido hablado se puede concebir como generador de una onda de sonido, o patrón de vibraciones, en el aire. Puesto que la mayor parte de los sonidos son bastante complejos, es conveniente pensar que cada expresión individual está compuesta por un número de distintos componentes, cada uno de ellos de diferente frecuencia, y en los que la intensidad del sonido puede ser distinta para diferentes frecuencias.

Si examinamos la distribución de las intensidades para todos los componentes de la frecuencia de un sonido fonético en particular, veremos que, debido a que existen frecuencias particulares que resuenan en la cavidad bucal (algo así como el efecto que produce Vd. en algunas buenas notas cuando canta en el baño), hay ciertos valores de frecuencia en los que se produce una alta intensidad de excitación. Estas frecuencias críticas se conocen como frecuencias *formantes* y, como veremos, pueden ser de gran ayuda a la hora de clasificar los sonidos. Además, en una aproximación simple al reconocimiento de fonemas, es posible pensar en las frecuencias formantes como unas posibles candidatas a ser los descriptores de patrón que andamos buscando. Puesto que para un fonema dado se encontrará normalmente una serie de formantes, se acostumbra a etiquetarlos para una mejor referencia, como F1, F2, F3,... y así sucesivamente, comenzando con la frecuencia más baja y siguiendo con las superiores.

UNA SOLUCION PARA EL RECONOCIMIENTO DE VOZ

Si nos fijamos en los sonidos de las vocales, veremos que el formante puede ser sumamente útil. Debido a la forma en que los órganos vocales producen los sonidos de las vocales adoptando una posición fija mientras vibran las cuerdas vocales, las frecuencias del formante son fijas también (la forma de la cavidad de resonancia no cambia).

Supongamos que tomamos un número de ejemplos de cada uno de los sonidos vocales a, e, i, o, u, y medimos las dos primeras frecuencias del formante (F_1 y F_2) para cada ejemplo. Descartaremos, de momento, los formantes más altos. Ahora podemos intentar utilizar la posición de estos formantes (es decir, las frecuencias en las que suceden las intensidades punta) como descriptores de patrón para describir el fonema. Por supuesto, es natural que dentro de una clase de fonemas veamos diferencias individuales, al igual que en el caso visual, un conjunto de ejemplos escritos a mano de la letra "A" será ligeramente diferente; pero examine algunos resultados típicos como los que se muestran en la figura 4.1. Podemos ver que si permitimos un pequeño grado de solapa-

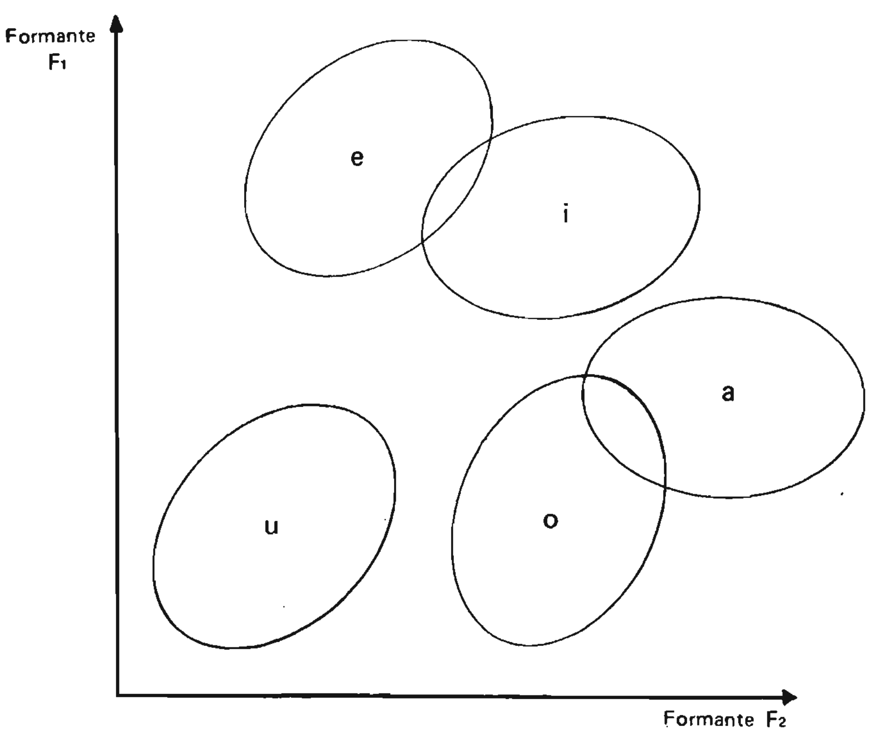


Fig. 4.1.—Relación entre las frecuencias formantes de las vocales.

miento, las clases de fonemas tienden a formar agrupaciones separadas e individuales en este espacio de rasgos, y por tanto, en principio, se podría encontrar un conjunto de funciones de decisión que podría utilizarse para identificar una expresión como miembro de una de estas clases. Así pues, para este caso sencillo, hemos especificado el problema de forma que corresponde exactamente a las situaciones abordadas en capítulos anteriores.

Desgraciadamente, cuando extendemos el problema para incluir un rango más amplio de fonemas y/o patrones de sonido más complejos, la situación se presenta algo menos clara. Una razón principal de esto es que en la producción de muchos sonidos los órganos vocales están realmente en movimiento mientras se produce el sonido. Este movimiento del aparato productor del sonido significa que la forma de la cavidad bucal también cambia, resultando en un cambio en la posición del formante. Si intentáramos utilizar la representación previa para tales casos,

donde midiéramos las dos primeras frecuencias formantes para utilizarlas como descriptores de patrón, tendríamos la situación mostrada en la figura 4.2, donde un patrón queda ahora definido por una traza o trayectoria de patrón como una función del tiempo a lo largo del espacio del patrón. Esto hace que la aplicación directa de nuestras anteriores técnicas sean un poco más problemáticas.

En realidad, una solución que podríamos intentar sería una toma de muestras a intervalos regulares de la duración de una onda de voz, quizás cada treinta milisegundos o algo así, ya que los cambios tienen lugar de forma bastante rápida, midiendo la primera y la segunda frecuencia formante en cada tiempo de muestreo. Esto podría facilitarnos la información para introducir el trazado de la trayectoria, tal como se ilustra y luego podríamos intentar comparar esta trayectoria con las trazas producidas por unos patrones de sonido arquetipo almacenados y definidos previamente. Por ejemplo, si estamos preparados para trabajar con un

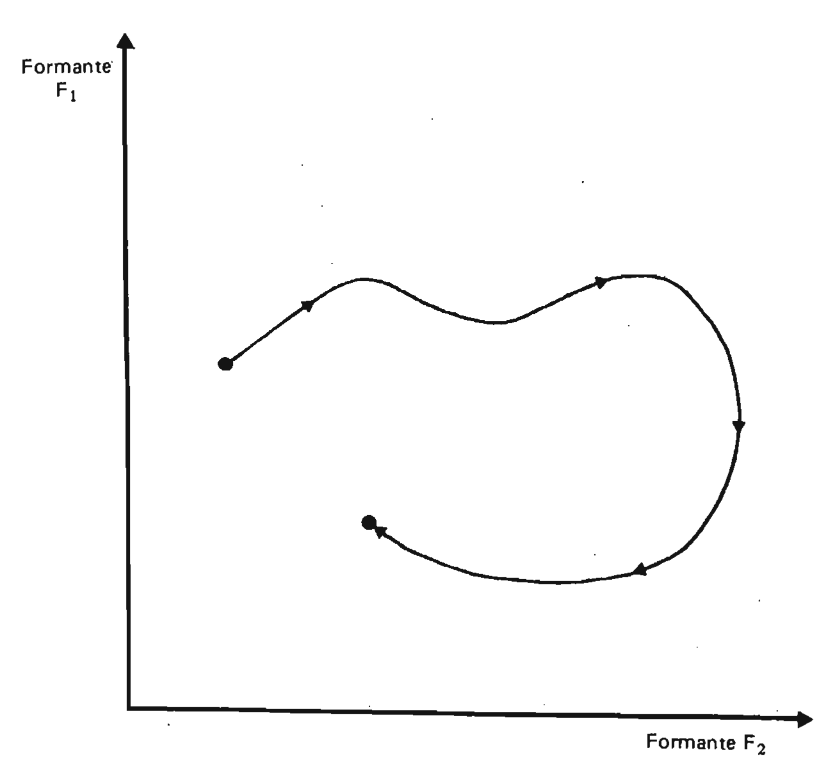


Fig. 4.2.—Cambios en las posiciones de las formantes.

vocabulario bien definido de palabras aisladas, por ejemplo, "COMIENZO", "PARADA", "IZQUIERDA", "DERECHA", "ARRIBA", "ABAJO", "RAPIDO", "LENTO", podemos intentar identificar un sonido comparando la traza que produce con las trazas arquetipos de todas las palabras conocidas del vocabulario escogido, tomando la mayor coincidencia entre ellos para identificar la expresión.

Verá que esta técnica, aunque es una solución al problema en cuestión, puede sufrir todas las dificultades que hemos encontrado anteriormente. Por ejemplo, podríamos esperar que si elegimos palabras similares en el vocabulario, la búsqueda de coincidencia será menos fiable y, se podría esperar una degradación progresiva según incrementamos el número de palabras diferentes que haya que distinguir.

Existen otros problemas, por supuesto, que surgen en gran parte debido a las diferencias individuales en las características físicas de los diferentes locutores, a la hora del día, a si el locutor está resfriado, etc. Quizás desee reflexionar un poco en cómo pronunciarían una misma palabra diferentes locutores (puede que incluso sean habitantes de distintas regiones del país), para demostrar así otro área de dificultad que se encuentra incluso en el problema restringido del reconocimiento de una palabra aislada.

No debemos omitir, en nuestra rápida inclusión al área del reconocimiento de voz, el hecho obvio de que los seres humanos inteligentes generalmente no conversan con palabras netas aisladas, sino que tienden a encadenar palabras para formar oraciones, frases e incluso fragmentos enteros de prosa.

Tan pronto como pasemos a los dominios de dicha voz ligada o continua, todas las dificultades que hemos discutido aparecen inmediatamente exageradas. Sería un ejercicio interesante para Vd., pensar con más detalle sobre estos grados de dificultad añadidos, pero aquí le expondremos algunos apuntes sobre las áreas conflictivas:

1. En la voz continua no aparece siempre claro donde hay que situar los límites entre las palabras, en gran parte debido a que nosotros tendemos a no establecer pausas a modo de puntos de señalización.

2. Al encadenar palabras para formar una expresión continua es inevitable que la articulación de un sonido quede afectada por el movimiento de los órganos vocales al cambiar de un fonema al siguiente. Como consecuencia, se obtienen sonidos que son versiones distorsionadas de los sonidos arquetipo oídos aisladamente.

3. Con frecuencia omitimos enteramente algunos fonemas, particularmente si su sonido es común al final de una palabra y el comienzo de la siguiente.

La mayoría de estos puntos, y otros, se pueden enfocar con más claridad si dedica algo de tiempo a pensar como sonaría la siguiente frase si fuera hablada de un modo conversacional más bien farragoso:

> "Todos los hombres son naturalmente buenos a menos que se demuestre lo contrario".

Imagine que oye esto a través de una línea telefónica y vea las ambigüedades que pueden surgir si Vd. tuviera que analizar el mensaje basado exclusivamente en los patrones de sonidos.

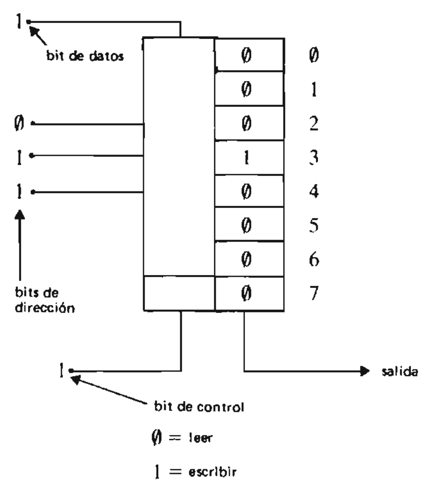
Esto nos lleva al punto final que es, tal y como ilustra el último párrafo, que con frecuencia podemos utilizar información adicional al idioma normal hablado para ayudar a analizar el contenido de un mensaje. Esta información adicional puede obtenerse utilizando las reglas gramaticales y estructurales de la lengua y el contexto real del mensaje que viene dado al considerar el significado de las palabras involucradas en él. Esta utilización de la información *sintáctica* y *semántica* puede aportarnos más confianza en la forma en que descifremos un mensaje hablado, pero la contrapartida es que el ordenador tiene que trabajar más para llevar a cabo el análisis que necesitamos hacer.

Finalmente, desde el momento en que llegamos a la etapa en la que consideramos el reconocimiento de voz en el contexto de pasajes de voz continua, o la interpretación de diálogos conversacionales, nos estamos acercando cada vez más al punto en el que nuestra distinción entre reconocimiento de voz y comprensión de voz se convierte de nuevo en algo importante y volvemos a las preguntas que nos hacíamos al comienzo de este capítulo. Estamos también en un punto en el que el área del *lenguaje* como un todo, en lugar de patrones de sonidos individuales, se convierte en el foco de atención; esto nos conducirá claramente, tras una corta pausa en los dos próximos capítulos, al área de la Comprensión del Lenguaje Natural, que es el tema del capítulo 7.

5
Aprendiendo mallas

Una de las puntualizaciones que hice en el capítulo 2 sobre las posibilidades del aprendizaje humano, se relacionaba con nuestra facilidad para generalizar. En este capítulo, examinaremos un modelo electrónico del proceso de aprendizaje en el que es fácil ver cómo ocurre el proceso de generalización.

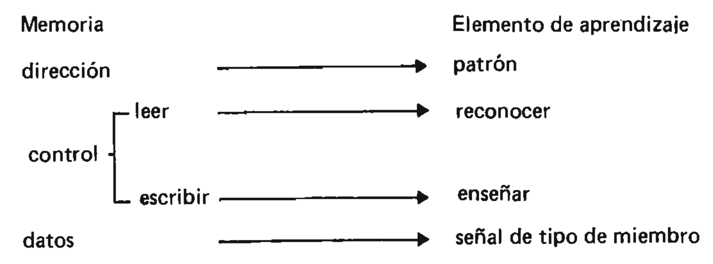
Primero necesitamos algunos conocimientos. Piense por un minuto en un fragmento de memoria de ordenador que consta de ocho bits en el que cada bit se puede direccionar por separado. Necesitamos pues tres bits de direccionamiento para poder identificar los ocho bits de datos numéricamente, del 0 al 7. También necesitamos un elemento de datos para almacenar (un sólo bit) y una señal de control que indique al sistema si debe escribir o leer un bit de la memoria. Por tanto, podríamos verlo así:



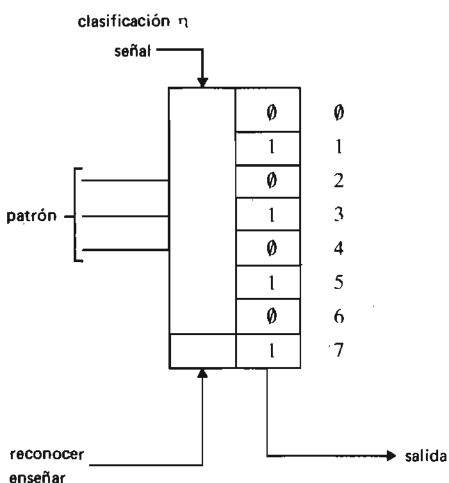
El ejemplo anterior va a escribir un "1" en el bit 3. El bit de control se pone a "1" para indicar que va a tener lugar una escritura, el dato a escribir es un "1" y los bits de dirección muestran un 3 binario.

Ahora, de forma un tanto primitiva, hemos "enseñado" a esta RAM de 8 bits a que reconozca el patrón 011, ya que si el bit de control se pone ahora a 0 (lectura) y toda la gama de patrones de bits (000 a 111) se presentan en las entradas de dirección una a una, el dispositivo sacará un "1" *solamente* cuando esté presente el patrón 011.

Por lo tanto a nuestros efectos, podríamos volver a designar los terminales del dispositivo de esta forma:

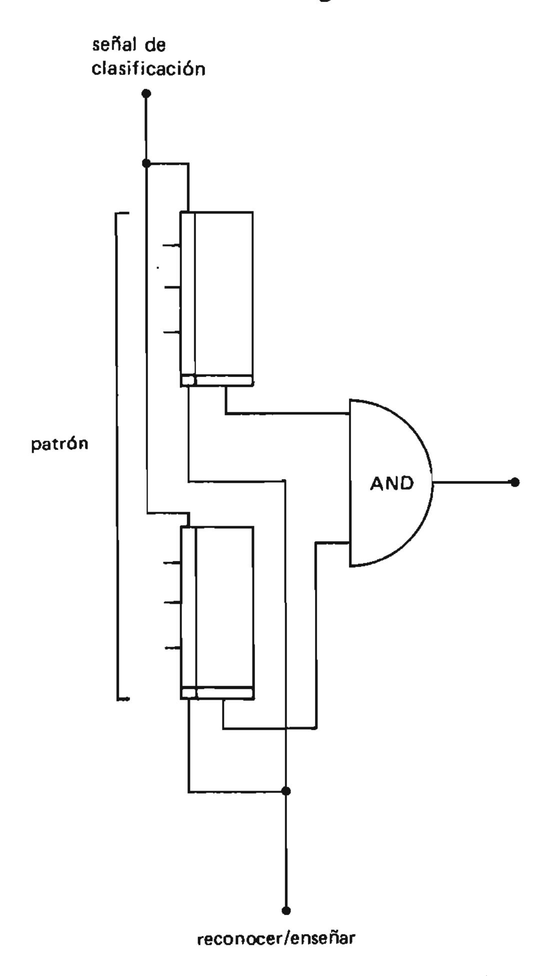


Por supuesto que varios patrones pueden considerarse como del mismo tipo, mientras que lo que se pretende que represente cada patrón puede variar. Podría ser un patrón visual o cualquier otra estructura representativa diferente. Si pensamos en que nuestro patrón de 3 bits representa enteros positivos, por ejemplo, podríamos enseñar al dispositivo a distinguir entre los números pares y los impares. Después de un ciclo completo de enseñanza, tendríamos:



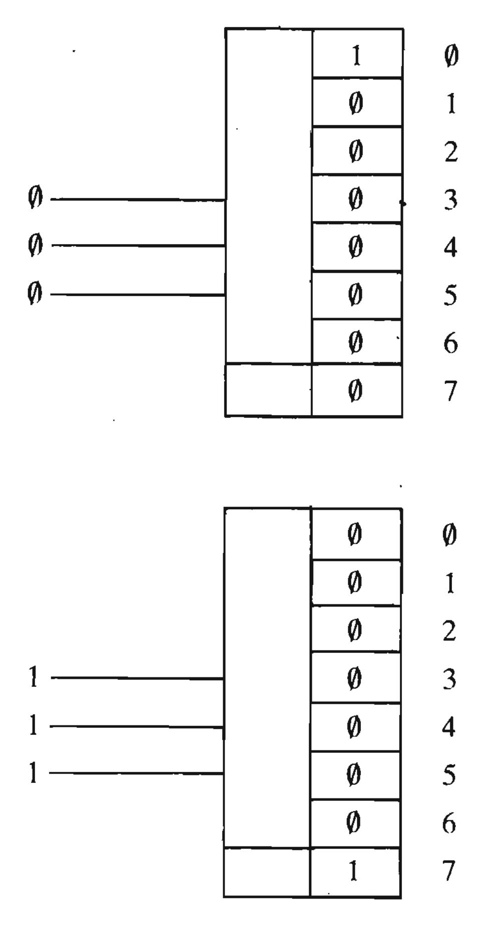
Hasta aquí todo esto no impresiona demasiado. Todo lo que hemos conseguido que haga el dispositivo es recordar algunas cosas (apenas nos sorprende, al fin y al cabo, estamos utilizando una memoria), y las vuelva a traer en respuesta a un mandato. Esto es un aprendizaje de memoria comparable a las tablas de multiplicación o un vocabulario de una lengua extranjera.

Sin embargo, supongamos que ahora queremos tratar patrones de 6 bits. Un método sería disponer de un elemento de memoria mayor (debería ser de 64 bits), pero una alternativa consistiría en dividir los patrones de 6 bits en dos partes de 3 bits y utilizar dos de nuestros elementos de memoria de 8 bits como sigue:



A las salidas de los dos elementos se les hace un "AND" conjuntamente. En otras palabras, la salida final es un "1" solamente si ambas salidas son un "1".

Supongamos que deseamos entrenar a esta malla para distinguir de nuevo entre números pares e impares. Comenzaremos presentando un 7, que es 000111:



Por tanto, la convención que estamos utilizando es que un número es par (representado por un cero en el bit apropiado) a menos que digamos al sistema que es impar poniendo la señal de clasificación a 1, en cuyo caso se pone a "1" un bit de cada elemento. Por supuesto, se asume que ambos elementos están inicializados a ceros.

Ahora continuaremos al proceso de "entrenamiento" introduciendo los números 29 (011101) y 33 (100001) con la señal de clasificación puesta a "1". Después de esto, nuestros dos elementos aparecerían así:

	1	Ø
	Ø	1
	Ø	2
	1	3
	1	4
	Ø	5
	Ø	6
	Ø	7

	Ø	Ø
	1	1
	Ø	2
	Ø	3
	Ø	4
	1	5
	Ø	6
	1	7

Descubramos ahora qué es lo que "sabe" nuestro dispositivo. Obviamente si introducimos 7, 29 ó 33 con el sistema en modo de reconocimiento, ambos elementos sacarán unos (1's); por tanto, la salida final será un "1". Es exactamente lo mismo que sucedió para el caso de un solo elemento. Sin embargo, supongamos que entramos 31 (011111). Esto pone un 3 sobre el elemento más alto (y el bit 3 es un "1") y un 7 sobre el elemento bajo (en el que el bit 7 es un "1"). Por tanto, la salida final es un "1" y el dispositivo ha reconocido un 31 como número impar sin habérselo tenido que decir.

En realidad, reconoce el 1, 5, 25, 31, 37 y 39, así como el conjunto original de valores de entrenamiento. Por lo tanto, a pesar de que no "comprende" totalmente el concepto de "un número impar", sí *conoce* más números impares de los que le dijimos; lo que ha hecho ha sido *generalizar*.

UN PROGRAMA SENCILLO DE APRENDIZAJE DE MALLA

Simulemos ahora esta estructura en un programa. Puesto que cada elemento de memoria consta de 8 bits, será conveniente utilizar un byte para cada uno. Así:

```
20 DIM n$(2)
```

realizará el trabajo de momento. Más tarde, habrá que cambiarla, ya que nuestras mallas de aprendizaje van a ser mucho mayores de 2 elementos.

Ahora tenemos que escribir tres rutinas:

```
Inicializar   (1000)
Enseñar       (1200)
Reconocer     (1400)
```

Los números entre paréntesis indican los números de línea en los que comienza cada rutina. Al agrupar la codificación, Vd. puede escribir:

```
GOSUB    1400
GOSUB    1000
```

y así sucesivamente, o puede insertar una línea 10:

```
10 LET inici = 1000: LET enseñar = 1200: LET recon = 1400
```

y luego escribir:

```
GOSUB recon
GOSUB inici
```

etcétera.

Utilizaré la última solución ya que es más legible*. Según presente nuevas rutinas, especificaré sus números de línea de comienzo y dejaré a su elección que las llame por nombre o por número. Por supuesto, si Vd. utiliza mi convención, tendrá que editar la línea 10 cada vez que se añada una nueva rutina y si no lo hace así, deberá sustituir mis nombres por números.

inicializar

Comenzaremos esta rutina con los razonamientos usuales que son los más sencillos:

```
1000  FOR p = 1 TO 2
1010  LET n$(p) = CHR$ 0
1020  NEXT p
1030  RETURN
```

Dos puntos a observar: primero, que puesto que estamos tratando con caracteres, tenemos que utilizar CHR$ para hacer la conversión. En segundo lugar, la utilización aquí de un bucle FOR, que se hace conveniente ya que, según apunté anteriormente, nuestras mallas serán normalmente de más de 2 elementos y, de este modo, solamente necesitaremos editar la línea 1000 para cambiar el tamaño.

enseñar

Comenzaremos con una descripción esquemática del algoritmo:

```
Tomar o leer el número y la señal de clasificación
Si señal = 0 entonces RETURN
Convertir número a binario (b$) (1600)
Seleccionar los 3 bits más bajos de b$
Poner un "1" en el bit apropiado del elemento inferior (1800)
Seleccionar los 3 bits más altos de b$
Poner un "1" en el bit apropiado del elemento superior
RETURN
```

---

* Para el BASIC de Microsoft se tendrían estas diferencias principales:
- El GOSUB tiene que llevar después un número de línea y *no un símbolo*.
- La función CHR$ lleva el argumento entre paréntesis.
- Para tomar fragmentos de una tira de caracteres emplear la función MID$.

(N. del Editor).

Los números entre paréntesis indican los números de línea de comienzo de las rutinas. La codificación es:

```
1200  INPUT "Entre número"; num
1210  INPUT "Clasificación" (0 = impar, 1 = par";señal
1220  IF señal = 0 THEN RETURN
1230  GOSUB bina
1240  LET s$ = b$(TO 3): LET p = 1
1250  GOSUB ponbit
1260  LET s$ = b$(4 TO): LET p = 2
1270  GOSUB ponbit
1280  RETURN
```

Esto sigue el algoritmo esquematizado con bastante aproximación. La subrutina *bina* en la línea 1600 devolverá una cadena binaria en b$. Dos subcadenas (bits 1 a 3 y 4 a 6) pasan a su vez a *ponbit* (línea 1800) que pone a 1 solamente el bit referenciado por s$. También p pasa a *ponbit* para indicar qué byte (1 ó 2) se va a actualizar.

bina

El método estándar para convertir de decimal a binario es el de dividir sucesivamente por 2 el número decimal, tomando el resto de cada etapa y sumándoselo a la cadena binaria de esta forma:

```
1600  LET b$ = ""
1610  IF num = 0 THEN GOSUB pad: RETURN
1620  LET int = INT(num/2)
1630  LET rem = num - 2 * int
1640  LET b$ = STR$ rem + b$
1650  LET num = int
1660  GOTO 1610
```

En el proceso se destruye *num*, pero como en lo único que estamos interesados es en su equivalente binario, no nos importa.

Observe que la nueva subrutina *pad. enseñar* espera que b$ tenga seis dígitos, por lo que *pad* añade los ceros necesarios:

```
1700  IF LEN b$> = 6 THEN RETURN
1710  LET b$ = "0" + b$
1720  GOTO 1700
```

El uso de "> =" en la línea 1700 indica que *pad* creará una cadena de longitud seis para nuestros fines, pero *bina* puede aún utilizarse para valores de *num* mayores de 63 y, en tal caso, creará simplemente una cadena con longitud mínima.

ponbit

Ahora nos encontramos con un pequeño problema. Imagine que a *ponbit* se le ha pasado la cadena 011, de forma que queremos poner el bit 3. Supongamos también que el byte destino es actualmente:

00110000

En este caso es relativamente más fácil, puesto que todo lo que se necesita es añadir 8 (que es un 2^3) al byte de destino. Sin embargo, si la cadena ha sido 100 no se requiere ninguna acción, ya que el bit 4 ya está puesto. Si añadiéramos 2^4 acabaría en:

01000000

lo cual es claramente erróneo.

La solución sencilla para el problema es la de ejecutar una operación lógica OR sobre el byte destino y el valor 2^3 (ó 2^4 ó lo que sea). Luego tenemos:

```
               00110000
OR con         00001000     (= 2³)
              =00111000
```

y

```
               00011000
OR con         00001000     (= 2³)
              =00011000
```

Ahora aquí esta el problema: el BASIC del Sinclair no combina cada par de bits de esta forma cuando Vd. utiliza su operación OR. En realidad hace algo extrañamente distinto. (Pruebe PRINT 36 OR 1 y PRINT 36 OR 0 y verá lo que quiero decir). Por supuesto, está pensado para su utilización en sentencias como:

```
IF a < 7 OR b = 12 THEN...
```

y puesto que evalúa expresiones condicionales como "a < 7" para 0 (falso) ó 1 (verdadero), no importa si no se tratan los otros valores de forma convencional.

Podríamos profundizar en esto de distintas formas, pero es obvio que puesto que las operaciones lógicas con bits de este tipo se suministran directamente en código Z80, una rutina en código de máquina es la solución más sencilla.

Si no está familiarizado con el código de máquina en un Spectrum, lo que sigue le puede parecer una receta de cocina, pero si Vd. quiere comprender realmente lo que hacemos, le sugiero que lea algún libro especializado.

Primero localice su RAMTOP...

Siempre que trate con rutinas en código de máquina deberá asignar espacio de memoria para ellas, declarando así ese espacio no disponible para BASIC. Es muy fácil:

```
1 CLEAR 31999
```

declara que cada byte de memoria cuya dirección exceda de 31999 quede fuera de los límites del BASIC. Así se dejan 600 bytes libres para el código de máquina en una máquina de 16K y más de 32K en una máquina de 48K. Si tiene una máquina de 48K probablemente desee seleccionar un valor más alto, pero he organizado las cosas de forma que el código funcione sin ninguna modificación en cualquiera de los modelos. El efecto exacto de CLEAR es el de alterar la variable RAMTOP del sistema, pero también hace otras cosas. Por ejemplo, limpia todas las variables; por lo tanto asegúrese de que ésta es la primera línea de su codificación.

Ahora, para la primera parte de nuestro camino, la codificación debe funcionar. La rutina *ponbit* pasará a la rutina en código de máquina los dos bytes a hacer un OR después de hacerles POKE a las posiciones 32000 y 32001. Esto significa que la codificación puede comenzar en la posición 32002. El resultado será devuelto por la función USR; por tanto *ponbit* aparece así:

```
1800  POKE 32000,2 ↑ VAL("BIN" + s$)
1810  POKE 32001,CODE n$(p)
1820  LET n$(p) = CHR$ USR or
1830  RETURN
```

Hay dos puntos a observar. Primero, en la línea 1800 el valor binario representado por s$ se convierte a un número. Desafortunadamente, Vd. no puede utilizar la función BIN con algo que no sea una cadena de ceros y unos. Por tanto, debe crear una cadena a partir de la cual VAL pueda generar un número. No olvide que el "BIN" entre comillas debe teclearse utilizando una pulsación de función única. (En otras palabras, no es válido teclear "B", "I", y "N" como letras separadas). En segundo lugar, yo he escrito "USR or" en la línea 1820, en lugar de "USR 32002" por razones de claridad. Por supuesto, habrá que corregir la línea 10 para incluir:

```
LET or = 32002
```

El código de máquina

Llegamos ahora a la rutina de código de máquina. He aquí una posibilidad:

```
LD    B,00      06 00
LD    HL, 7D00  21 00 7D
LD    A, (HL)   7E
INC   HL        23
OR    A, (HL)   B6
LDC,  A         4F
RET             C9
```

Es bastante directo. HL se utiliza como apuntador para los dos bytes de datos (7D00 hexadecimal es 32000 decimal). El registro A se carga con el primer byte y hecho un OR con el segundo, habiendo incrementado HL. El resultado se sitúa en el registro C y como B se limpió al comienzo, el par BC contiene el resultado. Esto está bien, ya que USR devuelve siempre el valor en BC. El código hexadecimal ensamblado se muestra a la derecha y Vd. podría cargarlo con algo como:

```
30 FOR p = 32002 TO 32011
35 READ byte
40 POKE p, byte
45 NEXT p
50 DATA 6,0,33,0,125,126,35,182,79,201
```

La sentencia DATA contiene los bytes en código de máquina convertidos a decimal.

Pruebas

Al fin tenemos algo que puede probarse. Pruebe esto:

```
90 GOSUB inici
100 GOSUB enseñar
110 LET num = CODE n$(1):GOSUB bina:PRINT b$
120 LET num = CODE n$(2):GOSUB bina:PRINT b$
130 GOTO 100
```

Esto le permitirá probar *enseñar* mostrando el estado actual de los dos bytes n$ (1) y n$ (2) en binario después de cada ejemplo de enseñanza. Utilice 7, 29 y 33 como números impares en el conjunto de enseñanza y obtendrá los patrones de bit mostrados en las páginas 50 y 51.

Reconocimiento

Ahora podemos escribir *recon*. El algoritmo esquematizado es:

```
Tomar o leer número
Convertir número a binario (b$)
Seleccionar los 3 bits más bajos de b$
Seleccionar el bit direccionado
Seleccionar los 3 bits más altos de b$
Seleccionar el bit direccionado
Si ambos bits direccionados son "1" el número es impar: RETURN
El número es par
RETURN
```

que se codifica así:

```
1400 INPUT "Introduzca número a reconocer"; num
1410 GOSUB bina
1420 LET s$ = b$(TO 3):LET p = 1
1430 GOSUB sacabit
1440 LET bit1 = bit
1450 LET s$ = b$(4 TO): LET p = 2
1460 GOSUB sacabit
1470 IF bit AND bit1 THEN PRINT "impar":RETURN
1480 PRINT "par"
1490 RETURN
```

sacabit (línea 2000 en adelante) debe ver si el bit direccionado es 0 ó 1. En realidad devuelve el valor apropiado en bit:

```
2000  POKE 32000,2 ↑ VAL("BIN" + s$)
2010  POKE 32001, CODE n$(p)
2020  LET bit = USR and
2030  RETURN
```

and es una rutina en código de máquina que hace el AND de los bytes 32000 y 32001. El valor con el que se ha hecho POKE en la 32000 es una máscara que permite solamente el paso del bit direccionado de n$(p) que está en 32001. Funciona así: supongamos que tenemos 0100 en s$. Luego queremos examinar el bit 4 de n$(p). 2 ↑ VAL ("BIN" + s$) es 2 ↑ 4 = 16. Por tanto, el valor hecho poke es:

```
00010000
```

al que se le hace un AND con un patrón desconocido:

```
a b c d e f g h
```

generando:

```
000d0000
```

Por tanto, el bit 4 es el único permitido y el resultado es cero o no cero con arreglo a si d era cero ó 1. La rutina en código de máquina es, por supuesto, muy semejante a la operación OR.

```
                          Hex            Decimal
32012    LD B,00          06   00           6,  0,
         LD HL, 7D00      21   00  7D      33,  0,   125
         LD A, (HL)       7E              126,
         INC HL           23               35,
         AND A, (HL)      A6              166,
         LD C, A          4F               79,
32021    RET              C9              201
```

Esta codificación se puede añadir a la sentencia DATA en la línea 50, en cuyo caso, por supuesto, la línea 30 debe editarse como:

```
30  FOR p = 32002 TO 32021
```

(No olvide añadir "LET and = 32012" a la línea 10).

Prueba final

Sustituya la línea 130:

```
130    INPUT "mode";m$
```

y continúe:

```
140    If m$ = "e" THEN GOTO 100
150    GOSUB recon
160    GOTO 130
```

Esto situará al sistema en modo "enseñar" para comenzar, pero consecuentemente Vd. podrá alternar entre el modo "enseñar" y el de "reconocer", entrando una "e" o una "r" respectivamente después del "modo" de mensaje de diálogo (prompt).

Verá que el sistema trabaja de forma muy parecida a lo que pronostica la teoría. Introduzca media docena de números impares al azar en modo "enseñar" y los reconocerá, y probablemente algunos más en modo "reconocer". Pero ¿tiene Vd. la impresión de que para algunos conjuntos de números de entrenamiento concluye "reconociendo" más números impares (o, si Vd. quiere, cometiendo menos erroes de reconocimiento) que para otros conjuntos, incluso cuando los conjuntos tienen el mismo tamaño? Bien, está Vd. en lo cierto si piensa que es así.

Minimizando el Conjunto de entrenamiento

Esto nos sugiere que hay que indicar al sistema unas pocas cosas para que comprenda completamente el concepto de "número impar", en el supuesto de que escojamos los ejemplos más sugestivos.

Puede ser instructivo examinar los patrones de bits creados al enseñar los números impares 1, 3, 5, 7 y así sucesivamente hasta el 63. Por supuesto, podríamos hacerlo utilizando el programa llamador en curso, pero sería más fácil sustituirlo por:

```
 90  GOSUB inici:LET señal = 1
 95  FOR q = 1 TO 63 STEP 2
 96  LET num = q
100  GOSUB enseñar + 20
110  LET num = CODE n$(1):GOSUB bina:PRINT b$
120  LET num = CODE n$(2):GOSUB bina:PRINT b$: TAB 28;q
125  NEXT q
```

y observe los patrones de bits al desdoblarse. Obtendrá esto:

000001	000010	1
000001	001010	3
000001	101010	5
000001	10101010	7
000011	10101010	9
000011	10101010	11

Aquí tenemos algo interesante. Las cinco primeras entradas han cambiado el patrón de bits y, si Vd. quiere, se han incorporado al almacenamiento del conocimiento. Pero la sexta (11) no puede haber hecho nada útil ya que el patrón de bits permanece inalterado. Por supuesto, si lo decimos de otra forma, significa que el sistema podría haber reconocido el 11 habiéndose entrenado con los cinco números precedentes. Continuemos:

000011	10101010	13
000011	10101010	15
000111	10101010	17

Así pues, el 13 y el 15 son también redundantes, pero el 17 altera algo. El próximo cambio tiene lugar en 25, luego en 33, 41, 49 y finalmente en 57. Por tanto, un conjunto mínimo de entenamiento aparecería así:

1, 3, 5, 7, 9, 17, 25, 33, 41, 49, 57

Incluso esto puede mejorarse. Para ver cómo, piense en el problema a la inversa. El patrón final de bits que aparece es:

11111111	10101010
a b c d	a b c d

Ahora, para comenzar, podemos seleccionar unos números que pongan un bit en cada byte. Por ejemplo, los dos bits señalizados con una "a" se ponen cuando al sistema se le ha "enseñado" 63. Los señalizados con una "b" se pondrán por el 53, los de "c" y "d" se relacionan con el 43 y 33 respectivamente. Luego necesitamos 4 números más para poner los restantes 4 bits: 25, 17, 9 y 1 se encargarán de ello. (Por supuesto, existen otras posibilidades; intente experimentarlas). Por tanto, necesitamos decir al sistema solamente ocho números impares para poder identificar los 32 del conjunto.

Otros conjuntos de entrenamiento

Recordemos que, aunque hemos seleccionado como ejemplo distinguir entre unos números pares e impares, no existe nada en el programa (excepto mensajes), que nos limite a estos tipos. Intente hacer que el programa reconozca por ejemplo, los números múltiplos de 4 + 1 (5, 9, 13, 17, etc.). Funciona bien ¿verdad?

Esto sugiere que Vd. podría hacer una modificación para que el sistema realice un pequeño juego para utilizarlo en una fiesta familiar. Haga que averigüe el siguiente número en una secuencia dada. Los números dados actúan como conjunto de entrenamiento. El programa deberá comprobar todos los números subsiguientes hasta encontrar uno que pueda reconocer y lo designe como número objetivo. ¡Así puede resolver un test típico de inteligencia!

Hasta ahora pensamos que disponemos de una estructura más bien mágica que es capaz de proezas notables de reconocimiento, pero sólo parcialmente porque soy culpable de cierta prestidigitación. He escogido ejemplos que son idóneos para reconocimiento. El procedimiento se sentirá totalmente confundido al intentar distinguir los números primos o los múltiplos de 3.

Sin embargo, tampoco podemos esperar que un cerebro de 2 bytes vaya a rivalizar con Einstein.

6
Mallas de aprendizaje y patrones visuales

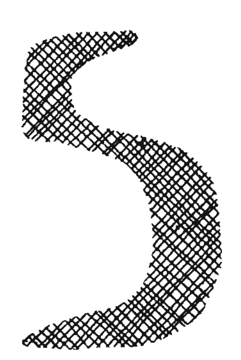
Nuestra primitiva malla de aprendizaje del capítulo 5 podría distinguir, de modo limitado, entre distintos tipos de números. Pero esto lo hacía de un modo completamente no numérico; de hecho, simplemente examinaba patrones de bits. Por tanto, no es un salto de gigante sugerir que debemos ser capacer de diseñar un sistema semejante para reconocer patrones visuales (dibujos y símbolos) en el supuesto caso de que puedan representarse como combinaciones de ceros y unos.

Esto es sencillo. Después de todo, una fotografía de un periódico consta de puntos (o ausencia de ellos), por lo que podemos utilizar un "1" para el "punto" y un "0" para el "espacio" de forma casi idéntica. En el caso de la fotografía impresa variando la densidad con la que están empaquetados los puntos se consigue una escala de grises.

Nosotros no podemos hacer esto; por tanto, nuestras "imágenes" serán en blanco y negro,(de momento, porque más tarde veremos como *se puede* representar una escala gris).

Vamos a ceñirnos a un pequeño conjunto de patrones que conste de los dígitos 1, 2, 3, 4, 5, 6, 7, 8 y 9.

El "5" aparecería así:



y en la "retina" de la máquina sería:

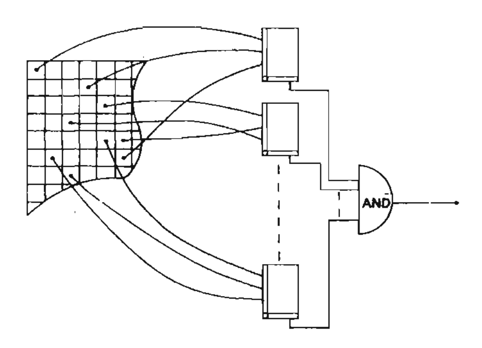
```
00000000
00001100
01111100
01100000
01111000
00001100
00001100
00001100
00011100
00111000
00000000
00000000
```

Lo que he llamado la retina es, por supuesto, una matriz (array) de dos dimensiones de ceros y unos. Ambas formas de la silueta están alargadas en comparación con un "5" típico escrito a mano, alargamiento que muestra irregularidades que normalmente están presentes, pero que ignoran (o se compensan) nuestros sistemas visuales. Por ejemplo, algunas partes de la línea son más gruesas que otras y la parte alta de la figura no es perfectamente horizontal. Sin embargo, esto no representa un problema por dos razones. La primera es que ya hemos visto que una malla de aprendizaje puede estar pensada para reconocer *tipos* de patrones; también veremos más tarde que existen técnicas de proceso de imágenes que "arreglan" una silueta antes de que intentemos reconocerla.

Ahora el patrón de la retina facilitará la entrada a nuestro conjunto de aprendizaje, que al igual que antes, constará de un conjunto de celdas de 1 byte. Cada celda tiene una entrada de 3 bits, por lo que necesitaremos 32 celdas en total. (Es decir, n$ debe redimensionarse a 32 de longitud).

Hasta ahora hemos utilizado la conexión natural entre el patrón y el número que representa para decidir la referencia ente los bits y las líneas de dirección de celda. Realmente, la única razón para hacer esto era la de facilitar la codificación. Las conexiones podrían haber sido al azar sin afectar por ello a las posibilidades de reconocimiento del dispositivo. (Sin embargo, las conexiones cambiadas habrían alterado las *clases* de cosas para las que el reconocimiento era bueno. Por ejemplo, otra configuración podría haber tratado mejor los múltiplos de 3, pero no sería tan idónea para números pares e impares).

Por tanto, en principio podríamos tener algo como:



Aunque ¿no vamos a tener problemas al modelar las conexiones de manera tan confusa? Sorprendentemente no y además es mejor hacerlo así, ya que podemos disponer de diferentes conexiones muy fácilmente y por tanto experimentar para encontrar las configuraciones ideales.

Necesitamos 2 matrices bidimensionales. La primera es la propia retina, que de momento hemos escogido de 8 × 12:

```
DIM r$(12,8)
```

La segunda es un conjunto de coordenadas de fila y columna que indica dónde encontrar cada elemento del dibujo ("pixels" o picture element) en orden:

```
DIM c(96,2)
```

Por ejemplo, c puede comenzar así:

c

7	2
3	6
4	4
6	5
8	1
9	6
:	:
:	:

Esto nos indica que los tres pixels que forman la dirección o patrón de entrada para la celda 1 se encuentran en r$ (7,2), r$ (3,6) y r$ (4,4) y que los de la celda 2 están en r$ (6,5), r$ (8,1) y r$ (9,6). Podemos inicializar c laboriosamente a mano, o por medio de alguna rutina al azar. En cualquier caso, nos ocuparemos de ello más adelante.

En este momento deberíamos comprobar la cantidad de memoria que se va a utilizar. Cada número ocupa 5 bytes; por tanto, la matriz de coordenadas es 96 × 2 × 5 = 960 bytes. Además, r$ es de 96 bytes y n$ será ahora de 32 bytes. Todavía estamos aproximadamente sobre 1K de memoria, lo cual no presenta problemas. Si deseamos incrementar más tarde el tamaño de la retina, será bueno no olvidar que c puede crearse como una matriz de caracteres, ya que no contiene nunca valores mayores de 255. Esto disminuiría el espacio necesario para la matriz en una quinta parte. De todas formas he hecho que la retina sea una matriz de caracteres por razones que pronto veremos justificadas.

De momento asumiremos que la entrada a la retina es tratada por una rutina llamada *retin*.

inici necesita cierta modificación marginal:

```
1000  FOR p = 1 TO 32
```

enseñar requiere más arreglos, pero la práctica totalidad de la estructura permanece igual:

```
1200  GOSUB retin
1210  INPUT "Clasificación (0 = non – 5,1 = 5)";señal
1220  IF señal = 0 THEN RETURN
1230  FOR p = 1 TO 32
1240  GOSUB tomabits
1250  GOSUB ponbit
1260  NEXT p
1270  RETURN
```

retin obtiene el patrón. El mecanismo de la señal de clasificación no se altera, excepto, por supuesto, que el tipo de objetivo es "cosas que parezcan un 5" o cualquier otra silueta que escoja.

Ahora necesitamos poner cada una de los 32 celdas. *Tomabits* es una rutina nueva para formar la dirección requerida seleccionando los bits apropiados de la retina y situándolos en s$ (ya que ahí es donde *ponbit* espera encontrarlos). *ponbit* no cambia, ya que se le pasa no sólo s$ sino también p, que le indica qué byte de n$ debe cambiar.

tomabits comienza en 2200:

```
2200  LET s$ = " "
2210  FOR b = 1 TO 3
2220  LET row = 3 * (p – 1) + b
2230  LET s$ = r$(c(row,1),c(row,2)) + s$
2240  NEXT b
2250  RETURN
```

Aquí hay un par de pequeños trucos. El primero es que la fila se calcula para decirnos qué filas de c vamos a tratar. Serían las filas 1, 2, 3 cuando p = 1; 4, 5 y 6 cuando p = 2 y así sucesivamente. La expresión de la línea 2220 genera estos valores.

El segundo es que en la línea 2230, s$ se construye gradualmente de igual manera que en *bina*, pero los subíndices de la matriz parecen un poco chapuceramente pensados. Es fácil verlo si piensa Vd. en un ejemplo:

Supongamos que b = 1, p = 2 y c se crea como se mostró previamente. Estamos interesados en este momento en la fila 3 * (2 – 1) + 1 = 4 que apunta al elemento de la retina correspondiente al bit más bajo del segundo byte de n$. Por tanto, estaríamos seleccionando r$ (6,5); c (4,1) es 6 y c (4,2) es 5.

retin (2400)

Eventualmente, probablemente deseemos una versión de *retin* que nos permita dibujar imágenes en alta resolución sobre la pantalla y cuyo resultado se dibujará luego sobre nuestro array de 8 × 12.

Sin embargo, tengo dos objecciones que hacer a la introducción de algo muy sofisticado, en este punto. La primera es que va a ser algo difícil. La segunda (y más seria) es que las pruebas del resto del sistema se harán más complejas de lo necesario. Si la representación externa de una imagen aparece como un bloque de 8 × 12, es fácil relacionarla con el array de 8 × 12 e imaginarnos unos patrones de pruebas apropiadas. Si la relación es más complicada, también lo será la creación de las pruebas. Por tanto, por el momento, *retin* hace exactamente dos cosas:

1. Permitir al usuario introducir un patrón en una retina limpia.
2. Permitir al usuario editar el patrón actual.

La opción 2 tiene un doble uso. El primero nos permite efectuar cambios en caso de error. El segundo nos permite generar una serie de

cuadros semejantes con un mínimo de dificultad, y de hecho vamos a querer tener una serie de cuadros semejantes para un conjunto de entrenamiento. He aquí el algoritmo:

1. GET opción (nuevo, editar o salir)
2. IF opción = s THEN RETURN
3. IF opción = n THEN limpiar r$: editar: GOTO 1
4. editar: GOTO 1

Pensando así en el problema, sólo existe una diferencia entre introducir un nuevo cuadro y hacer una edición (edit). En el primer caso se limpia primero la retina y luego se edita. Así nos ahorraríamos algo de trabajo. He aquí la codificación:

```
2400  INPUT "nuevo, editar o salir"; o$
2410  IF o$ <> "n" AND o$ <> "e" AND o$ <> "s" THEN GOTO
      2400
2420  IF o$ = "s" THEN RETURN
2430  IF o$ = "n" THEN GOSUB limpia:GOSUB editar:
      GOTO 2400
2440  GOSUB editar
2450  GOTO 2400
```

limpia (2600)

Esto es fácil:

```
2600  FOR y = 1 TO 12
2610  LET r$(y) = "00000000"
2620  NEXT y
2630  RETURN
```

editar (2800)

Aquí el primer trabajo es el de mostrar el estado actual de r$. Utilizaré un tablero de ajedrez con cuadros color verde y cyan para indicar una retina limpia y luego sobreescribiré en negro para indicar un correspondiente "1" en r$. Por supuesto, Vd. podría tener un papel de un solo color, pero esto dificultaría algo más el averiguar cuál es cada celda:

```
2800  FOR y = 1 TO 12 STEP 2
2810  FOR x = 1 TO 8 STEP 2
```

```
2820 PRINT PAPER 4; AT y,x;" "; AT y + 1,x + 1;" "
2830 PRINT PAPER 5; AT y,x + 1;" "; AT y + 1,x;" "
2840 NEXT x
2850 NEXT y
2860 GOSUB copiar
```

Probablemente esté pensando en algo más claro que las líneas 2820-2830 pero ¡es difícil escribirlo! *Copiar* tomará el patrón actual de unos (1) en r$ y le mostrará como cuadros negros en la pantalla.

Ahora las características de edición. Cada celda se puede editar comenzando a partir del ángulo superior izquierdo. La celda que se puede editar actualmente, contendrá un signo "#" actuando como cursor. Las teclas de cursor (sin SHIFT) permitirán que el cursor se desplace a voluntad. Un *editar* consistirá en la simple pulsación de la tecla EDIT (de nuevo, sin SHIFT), que combinará el estado de la celda marcada (es decir, de plano de fondo a negro o de negro a plano de fondo). Se puede abandonar el modo *editar* pulsando ENTER:

```
2870 LET x = 1:LET y = 1
2880 PRINT OVER 1;AT y,x;"#"
2890 LET c$ = INKEY$
2900 IF c$ = "" THEN GOTO 2890
2905 FOR q = 1 TO 20:NEXT q
2910 IF CODE c$ = 13 THEN RETURN
2920 IF c$ = "1" THEN GOSUB lanzar
2930 IF c$ = "5" AND x > 1 THEN PRINT OVER1;
     AT y,x;"#":LET x = x - 1:GOTO 2880
2940 IF c$ = "6" AND y < 12 THEN PRINT OVER1;
     AT y,x;"#":LET y = y + 1:GOTO 2880
2950 IF c$ = "7" AND y > 1 THEN PRINT OVER1;
     AT y,x;"#":LET y = y - 1:GOTO 2880
2960 IF c$ = "8" AND x < 8 THEN PRINT OVER1;
     AT y,x;"#":LET x = x + 1:GOTO 2880
2970 GOTO 2880
```

Hay uno o dos puntos a observar aquí. En primer lugar, el símbolo "#" se mueve enteramente en modo "OVER1", de forma que restaura el contenido previo de cada celda después de que se ha movido. Sin embargo, estamos utilizando en realidad tres colores de papel y el BASIC no puede tratar esto. Por tanto, cuando el cursor pasa sobre una celda vacía, se vuelve blanca. No encuentro en esto una incomodidad especial pero, si piensa que le merece la pena, Vd. podría jugar con el fichero de atributos para restaurar el papel verde y cyan. Las rutinas *and* y *or* le

ayudarán en esta conexión. En segundo lugar, el bucle nulo FOR que se ejecuta después de leer una tecla pulsada (línea 2905) facilita suficiente demora para que Vd. pueda levantar el dedo de la tecla pulsada sin que ocurra una segunda respuesta. El problema es que INKEY$ responde a los *cambios* en el estado del teclado, por lo que si no tiene cuidado ¡obtendrá una respuesta cuando pulse la tecla y otra cuando la libere! Si utiliza PAUSE para facilitar esta demora, piénselo con cuidado; no olvide que PAUSE queda desactivada al pulsar cualquier tecla.

copiar (3000)

No hay problema con esta rutina:

```
3000  FOR y = 1 TO 12
3010  FOR x = 1 TO 8
3020  IF r$ (y,x) = "1" THEN PRINT AT y,x;"■"
3030  NEXT x
3040  NEXT y
3050  RETURN
```

lanzar (3200)

Lo único que hace *lanzar* es cambiar un "1" en r$ por un "0" y viceversa. Ahora si 0 y 1 fueran los únicos números, podría escribir:

```
LET r (y, x) = NOT r (y, x)
```

ya que 1 y 0 representan "verdadero" y "falso".

Pero como son caracteres, si quiero hacer ese truco tengo que convertir el carácter en número (utilizando VAL), hacer un NOT al resultado e invertir el proceso con STR$:

```
3200  LET r$(y,x) = STR$ NOT VAL r$(y,x)
3210  IF r$(y,x) = "1" THEN PRINT AT y,x;"■"
3220  IF r$(y,x) = "0" THEN PRINT AT y,x;" "
3230  RETURN
```

Algunos cabos sueltos

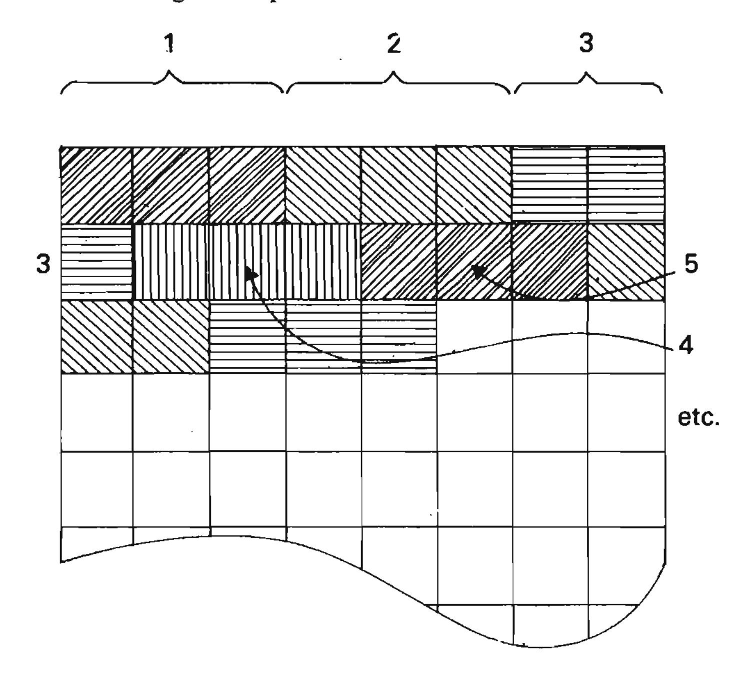
Existe una rutina diferida en el tiempo, cuya función es la de inicializar el array de coordenadas, c. La llamaremos *ponc* y comienza en la línea 3400. La organización más sencilla posible para c es:

1	1
1	2
1	3
1	4
1	5
1	6
1	7
1	8
2	1
2	2
2	3
2	4
2	5
2	6
2	7
2	8

12	5
12	6
12	7
12	8

En otras palabras, el primer grupo de tres ocupa la fila 1, columnas 1, 2 y 3; el siguiente grupo está en la fila 1, columnas 4, 5 y 6; el siguiente tiene tiene sus dos primeros elementos en la fila 1, columnas 7 y 8; pero su tercer elemento está en la fila 2 columna 1 y así sucesivamente.

En forma de diagrama aparecería así:



Adoptando mi habitual filosofía (no complicar las cosas hasta haberlas simplificado), utilizaremos esta disposición para comenzar:

```
3400  FOR y = 1 TO 12
3410  FOR x = 1 TO 8
3420  LET c(8 * (y – 1) + x,1) = y
3430  LET c(8 * (y – 1) + x,2) = x
3440  NEXT x
3450  NEXT y
3460  RETURN
```

Ahora editamos la línea 90 de la rutina para:

```
90  GOSUB ponc : GOSUB inici
```

y la nueva versión de *enseñar* está en un estado de prueba.

Reconocimiento

Ahora tenemos que ejecutar en *recon* los mismos tipos de revisiones. De nuevo, su formato general permanece invariable y, por supuesto, no hay que preocuparse de ninguna rutina principal como la *retin*.

```
1400 CLS:LET tot = 0
1410 PRINT AT 20,0;"Introduzca la figura a reconocer"
1420 GOSUB retin
1430 FOR p = 1 TO 32
1440 GOSUB tomabits
1450 GOSUB sacabit
1460 IF bit THEN LET tot = tot + 1
1470 NEXT p
1480 IF tot = 32 THEN PRINT "5"
1490 RETURN
```

Utilizamos *retin* para introducir una figura como antes. Utilizamos también *tomabits* para formar la dirección apropiada para cada uno de los 32 campos de tres bits y *sacabit* para averiguar si apunta a un "0" o un "1". No olvide, sin embargo, que *sacabit* devuelve un cero o un número positivo arbitrario para indicar "1"; por tanto sumando el bit a *tot* no nos dirá cuántas respuestas "1" hay. Deberemos comprobar *bit* y si no es cero, sumar 1 a *tot*. Hago especial hincapié en esto porque yo caí *exactamente* en esta misma trampa la primera vez que probé esta rutina y luego quedé asombrado preguntándome porqué tot había llegado a 332 cuando el bucle solamente se había ejecutado 32 veces.

Finalmente, debemos verificar tot para ver si todos los elementos (32) han respondido "1" y si ha sido así, imprimir un mensaje indicando que el patrón de prueba es un miembro del tipo de figuras que tratamos de identificar.

Pruebas

Las líneas 110 y 120 de la rutina llamadora no nos van a servir ya mucho más, pero por otro lado tampoco será necesario ningún cambio extra.

Una primera prueba sencilla es la de enseñar al sistema un único patrón, dejarlo en r$ (no editándola) y comprobar que se reconoce correctamente. Luego, haga un único *edit* (editar), es decir, cambie un elemento de la retina e intente reconocer el resultado. Por supuesto no será identificado como miembro de la clase del objetivo destino, pero lo

importante es que como sólo ha cambiado un pixel, 31 de las 32 celdas responderán "1", por lo cual tot deberá ser 31.

Una vez completada esta simple comprobación, puede probar con algunas más convincentes. Al cambiar dos pixels suficientemente distanciados, el tot bajará a 30. Sin embargo, si los dos pixels son miembros del mismo bloque de tres, el sistema identificará un solo cambio y tot quedará con 31.

Cambios en la representación de la retina

Es por esto por lo que fue conveniente comenzar con una representación tan directa; es fácil decir qué pixels son miembros del mismo bloque o no. Sin embargo, ahora podemos combinarlo si queremos.

En el array c está cada par de coordenadas, por tanto lo único que necesitamos hacer es volver a disponer su orden. Podríamos escribir una rutina llamada *desordenar*, para hacer esto:

```
3600  FOR n = 1 TO 100
3610  LET p1 = INT(RND * 96) + 1
3620  LET p2 = INT(RND * 96) + 1
3630  LET t1 = c(p1,1):LET t2 = c(p1,2)
3640  LET c(p1,1) = c(p2,1):LET c(p1,2) = c(p2,2)
3650  LET c(p2,1) = t1:LET c(p2,2) = t2
3660  NEXT n
3670  RETURN
```

Este algoritmo es muy común, si bien algo ineficaz. En cada bucle se crean dos apuntadores al azar, p1 y p2, y se intercambian los contenidos de las filas correspondientes de c. Si se repite esto 100 veces, significa que se han accedido a 200 filas aunque, por supuesto, se puede acceder a la misma fila varias veces y por término medio podemos esperar que se acceda dos veces a cada fila.

Naturalmente, no hay problema en repetir esta operación cada vez que se ejecuta el programa; por tanto, tendría sentido hacer:

```
GOSUB ponc : GOSUB desordenar
```

como mandato directo y luego ejecutar el programa desde la línea 90 (pero ¡quite primero de aquí el GOSUB ponc!). Cuando Vd. guarde el resultado, se preservarán los arrays o tablas, por supuesto.

Proyectos

1. Estamos entrenando al sistema con una figura en particular. En otras palabras, el programa nos permite decir "Esto es un 5" y actuará sobre esa sentencia. Si decimos "Esto no es un 5" el programa no toma acción ninguna. Revíselo de forma que escriba un cero en el bit apropiado cuando se le muestre una figura que *no* es miembro de la clase objetivo.

2. Lo que tenemos por el momento es un sistema que distinguirá un clase de figuras de otras. Su modificación para que pueda distinguir entre varias clases no presenta serios problemas. La señal de clasificación se convierte en un array o vector en lugar de una única variable (por tanto, si estuviéramos enseñando un "6" al dispositivo, f(6) se pone a "1" y así sucesivamente); n$ se convierte en un array 2-D, ya que existirá un "diseño" de 32 bytes para cada una de las clases reconocibles; y eso cambia *ponbit* y *sacabit*. También cambia *recon* porque tiene que examinar cada diseño de n$ para encontrar una coincidencia.

3. La prueba que propuse anteriormente de cambiar un solo pixel y asegurar que tot se reduce en uno exactamente, sugiere un camino interesante. Si no se encuentra una coincidencia exacta, atenúe ligeramente las condiciones para la coincidencia (es decir, acepte tot > 30, por ejemplo). ¿Esto nos conducirá a un número mayor de errores de reconocimiento? ¿O simplemente mejora las características de una generalización?

7
La comprensión del lenguaje natural

Durante casi cuarenta años los científicos relacionados con los ordenadores han dedicado un gran esfuerzo al diseño de lenguajes artificiales con los que dialogar con los ordenadores. Primero vinieron los códigos ensambladores y de máquina, lenguajes dificultosos en el sentido convencional. Luego, a mediados de la década de los 50, apareció el FORTRAN seguido rápidamente por el ALGOL60, COBOL, BASIC, LISP y PL/1. Después de una pausa vinieron APL, Pascal, C y más recientemente Prolog, Ada y Occam. Esto no constituye, en absoluto, una relación exhaustiva: existen literalmente cientos de lenguajes de ordenador que se utilizan actualmente. Existen además cientos de ellos más que han florecido de forma breve, quizás en un ambiente limitado, antes de ser sustituidos por otra alternativa más conveniente.

Por tanto, ¿por qué ha sido necesaria esta proliferación de lenguajes de ordenador? ¿Por qué no se ha desarrollado un lenguaje de ordenador multi-propósito? O incluso mejor, ¿por qué no se canalizó todo este esfuerzo para lograr que el ordenador comprendiera los mandatos escritos en idioma inglés ordinario? (o francés o castellano, etc.).

Veamos una a una todas estas preguntas. Primeramente, se han realizado intentos para diseñar lenguajes multi-propósito. Hoy día, generalmente están considerados como de difícil manejo. Su propia naturaleza cerrada les hacen difíciles de recordar, menos aún la utilización correcta de todas sus características. Los lenguajes modernos multi-propósito tienden a tener pocas funciones intrínsecas, pero permiten al usuario escribir las suyas propias y encadenarlas fácilmente con el cuerpo del lenguaje, para hacerle crecer si así lo desea. La última solución tiene, sin embargo, un inconveniente en sí misma. Significa que se deben escribir toda una serie de utilidades, clasificaciones, procedimientos de búsqueda, etc. No son *parte* esencial del lenguaje. En cierto sentido, tenemos algún tipo de código de máquina muy potente.

Por tanto, la visión moderna es que es más una cuestión de disponer de caballos para una carrera que de que exista un consenso sobre qué caballo para qué carrera. Si desea suscitar una controversia, seleccione simplemente cualquier problema que no sea trivial y pregunte a los científicos de ordenador qué lenguaje escogerían para codificarlo.

Lo que surge, pues, es una alternativa, la de utilizar un lenguaje natural. Piense en las tres sentencias siguientes:

1. Calcular la suma de 17 y 38.
2. Quizás desearía sumar 17 a 38.
3. ¿Cuánto es 17 más 38?

Para nosotros las tres tienen idéntico significado, aunque en el segundo ejemplo *está* claramente encubierto un mandato. Incluso las palabras utilizadas y la forma en que se presentan, difieren drásticamente en cada caso. En realidad, con excepción de los números, sólo hay una palabra común a dos frases e incluso ésta no aparece en la tercera. En BASIC tendríamos que escribir:

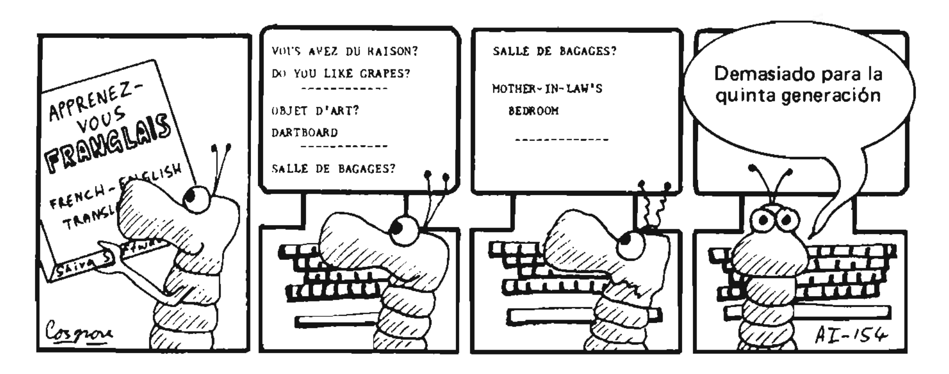
```
PRINT 17 + 38
```

Pero, existen cambios marginales posibles:

```
PRINT 38 + 17
```

o:

```
LET A = 38: LET b = 17: PRINT a + b
```



pero en todos los casos la palabra PRINT y la operación "+" son elementos centrales de las posibilidades BASIC para interpretar y ejecutar el mandato.

Es la asombrosa flexibilidad de un lenguaje natural la que le confiere su potencia y es esa flexibilidad la que aporta los mayores dolores de cabeza al diseñador potencial de un sistema de entendimiento o comprensión de un lenguaje natural.

Empleemos algo más de tiempo en la costa tratando de encontrar algunas conchas marinas antes de adentrarnos en las aguas profundas.

SINTAXIS Y SEMANTICA

Existe una estructura gramatical en cada lenguaje natural que determina las categorías de las palabras: sustantivos, verbos, adjetivos, etc., el orden en el que pueden aparecer y las terminaciones a aplicar a la raíces de las palabras en casos específicos. Por ejemplo, en castellano sabemos que una frase como "Tú ve el gato" es incorrecta gramaticalmente (o sintácticamente) ya que cuando se utiliza ese verbo en segunda persona del singular hay que añadirle una "s". Análogamente, "Tú el gato ves" no es válida, porque el verbo aparece al final de la frase, lo cual sucede en castellano en contadas ocasiones (aunque en alemán es algo muy normal).

Es interesante observar que estos errores no afectan a nuestra posibilidad de comprender lo que significan ambas sentencias. Por tanto, debe existir algún otro mecanismo que opere junto con la sintaxis. Es lo que llamamos la semántica. Es realmente otra palabra para el significado.

Al igual que se puede tener una frase sintácticamente incorrecta sin destruir su significado, podemos inventar frases con una sintaxis perfecta pero sin valor semántico, alguno. Por ejemplo:

Los conceptos azules se desvían sin fin

y

Ves a ramar un moreto berco.

En el primer caso, cada palabra nos es familiar, pero aunque la frase es innegablemente perfecta en el sentido gramatical, no conlleva significado alguno. No nos imaginamos que los conceptos tengan color alguno como atributos y ¿qué puede significar que "los conceptos se desvían"? En el segundo ejemplo, la mayoría de las palabras no tienen significado

alguno. Esto no nos detiene para establecer la estructura que tiene probablemente la frase. "ramar" parece ser un verbo; "berco" un adjetivo y "moreto" un sustantivo. Si yo le dijera:

Ves a ramar un moreto bercosamente

Vd. me podría decir "No tengo ni la más vaga idea de lo que está Vd. hablando, pero ha puesto un adverbio donde deber estar un adjetivo". Estraño ¿verdad?

La realidad es que si no tuviéramos esta especial habilidad o intuición, seríamos incapaces de mejorar nuestro vocabulario o incluso aprender el lenguaje en nuestra infancia. Después de todo, la alternativa es rehusar obstinadamente a involucrarnos con una frase que incluye una palabra nueva para nosotros, tal como procedería un intérprete o un compilador de lenguaje. Por lo tanto, si utilizo una palabra que no está en su vocabulario, por ejemplo "mesa", Vd. dira "No comprendo la palabra mesa" y yo respondo "Una mesa es una estructura de madera con cuatro patas que soportan una superficie plana" y Vd. dice "No comprendo la palabra mesa". ¿No es frustrante?

Así pues, parece como si las personas analizaran la sintaxis y la semántica de una sentencia al unísono. Cada parte aporta pistas a la otra. Podemos verlo muy claramente en mi ejemplo preferido:

La fruta baja del árbol.

Podemos considerar un sujeto (la fruta), un verbo (baja) y una circunstancia (del árbol). Sin embargo, se podría haber considerado una estructura de sintaxis menos obvia. Supongamos que "fruta" es un sustantivo y que "baja" es un calificativo. Ahora la frase se podría pensar con el significado de:

la fruta (de la parte) baja del árbol.

Esta interpretación en realidad produce confusión. Por tanto, en este caso, tenemos una estructura sintáctica ambigua que conduce a un valor semántico ambiguo para la frase.

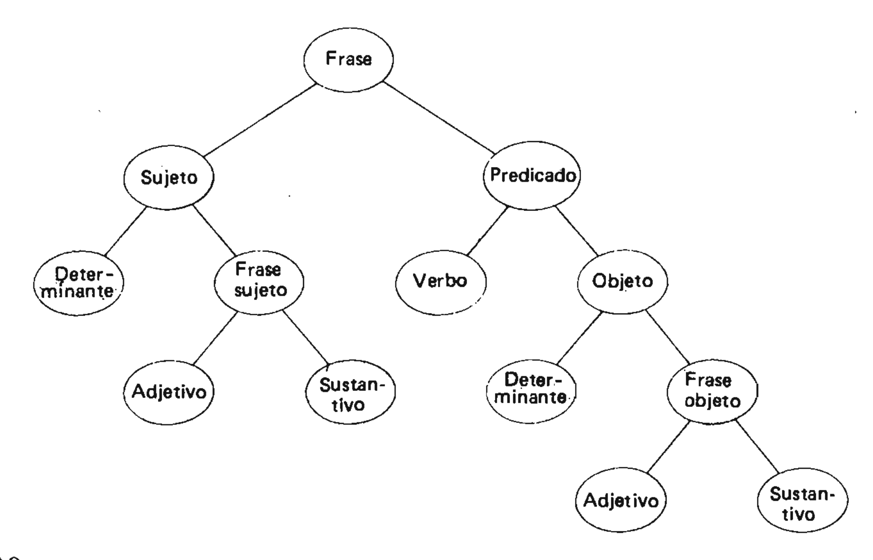
Si todo esto parece que no nos conduce a ninguna parte, tampoco lo pretendemos en este momento. Estoy intentando sencillamente señalar algunas de las dificultades con las que se encuentran los diseñadores de sistemas de lenguaje natural y pienso que, colectivamente, estos ejemplos indican la razón de porqué tales sistemas son tan delicados en el terreno práctico.

De hecho, algunas de las primeras investigaciones dentro del campo del proceso del lenguaje natural, se llevaron a cabo en el ámbito de la traducción del lenguaje, no de su comprensión. Esto no sorprende demasiado si considera que los ordenadores comenzaron a aparecer justo en el momento en que organismos como las Naciones Unidas y la Comunidad Ecónomica Europea vislumbraron la necesidad de traducir miles de documentos a docenas de lenguajes diferentes. Obviamente era muy tentador intentar diseñar sistemas de traducción. Fracasaron estrepitosamente por las razones que hemos expuesto. Existen algunas historias bien conocidas, tales como la de un sistema de traducción inglés/ruso, ruso/inglés que se probó alimentándolo con proverbios ingleses, pasándolos a ruso, utilizando el sistema para su traducción de nuevo al inglés, comparando posteriormente el original con el resultado. Este resultado fue que ciertas frases originales inglesas volvían convertidas en frases absolutamente confusas, distintas y sin sentido.

ARBOLES SINTACTICOS

Demasiadas dificultades. Comencemos a intentar hacer algo. Es evidente, a la vista de la discusión anterior, que será más fácil pensar en la sintaxis de una frase que en su valor semántico; por tanto, de aquí partiremos.

Se puede utilizar una estructura tipo árbol para construir formas sintácticas bastante bien. He aquí un ejemplo sencillo.



Imaginemos ahora que disponemos también de un vocabulario muy limitado:

Determinantes	*Sustantivos*	*Adjetivos*	*Verbos*
un	maíz	grande	comer
una	elefante/s	tostado/a	come
el			comió
algún			

Una frase como:

El elefante grande comió el maíz tostado

es coherente con el vocabulario y con el árbol sintáctico. Podríamos analizarla buscando la ocurrencia de izquierda a derecha, de un verbo conocido, en este caso "comió". Lo que precede al verbo forma el sujeto y está formado por un determinante válido (el), un adjetivo (grande) y el sustantivo (elefante). Lo que sigue a continuación es el objeto que puede descomponerse de forma semejante en palabras. Incluso con una gramática tan reducida y con tan corto vocabulario, sería posible inventar frases prácticamente sin sentido. Un ejemplo, obvio puede ser "El maíz grande comió un elefante".

Lo que necesita realmente un sistema de comprensión del lenguaje natural es un tipo de "construcción universal" o base de conocimientos, si lo prefiere, que le indique que "los elefantes comen cosas pero el maíz no". No voy a intentar aún abordar ese problema, por tanto vamos a escamotearlo con un pequeño juego de manos.

ELIZA

A mediados de los años 60 un científico de ordenadores americano llamado Joseph Weizenbaum desarrolló un sistema convencional del lenguaje natural que atacaba el problema de la representación del conocimiento de forma clara. Utilizó los conocimientos de sus colaboradores humanos y simplemente los reflejó sobre él mismo.

El programa se llamó ELIZA por Eliza Doolittle, la protegida del profesor Higgins en la obra "Pigmalión" de George Bernard Shaw, (más conocida como "My Fair Lady").

Un diálogo típico con Eliza podría comenzar:

Alguien: Hoy me siento algo cansado.

Eliza: ¿Por qué supones que te sientes hoy algo cansado?

Alguien: He pasado mala noche.

Eliza: Entonces has pasado mala noche.

Alguién: Sí.

Eliza: Cuéntame algo más sobre tu mala noche.

¿Ve Vd. lo que quiero decir con "conocimientos reflejados"? Eliza está analizando simplemente las frases que aparecen, de forma sintáctica, y luego recomponiéndolas en una afirmación o una pregunta, cambiando el "yo" por el "tú", "mi" por "tu", "siento" por "sientes" y así sucesivamente.

Eliza no tiene ni remota idea de lo que significa "descansado" o que está relacionado con el insomnio, o que "pasar una mala noche" es otra forma de decir "pasar una noche en vela". Ni siquiera sabe que lo común es dormir por la noche. Las personas utilizan una cantidad extraordinaria de conocimientos acumulados para comprender incluso el sentido de una conversación tan trivial como ésta.

Sin embargo, desde un punto de vista externo, Eliza llevaría a cabo, de forma notable, el test de Turing. Puede ser engañoso para quien no conozca como funciona. Por ejemplo, a un amigo mío sin relación con los ordenadores le mostré cierto día una implantación del programa llamado Analiza II. Por espacio de veinte minutos mantuvo con el programa una conversación aparentemente sensata, si bien superficial y quedó altamente impresionado. Sin embargo, el tema, por alguna razón, se tornó en las actividades del servicio secreto y mi amigo tecleó:

Yo creó que tú eres un miembro de la Compañía TO YO SA

Analiza respondió:

¿Por qué crees que yo soy un miembro de la Compañía To tu SA?

Por supuesto así concluyó al instante el juego. Analiza no pudo distinguir las iniciales de la compañía al ir separadas por espacios. Lógicamente no siempre puede Vd. asombrar a todo el mundo.

Quien haya jugado con Eliza durante bastante tiempo no podrá evitar andar escuchando las conversaciones en los bares, autobuses o supermercados y pensar "¿No podría yo haber mantenido con Eliza esa charla?" Es asombroso observar cuantas conversaciones humanas se asemejan a las de Eliza. Es como si la gente se pusiera en "modo Eliza" cuando no están realmente interesados en lo que se está diciendo, pero piensan que las respuestas son necesarias, aunque sólo sea por simple cortesía.

ANALISIS DE LA SINTAXIS

Voy a presentar una implantación tipo Eliza muy primitiva. La primera tarea va a consistir en analizar una frase de entrada sintácticamente; por tanto dedicaremos nuestro esfuerzo a este problema exclusivamente.

En primer lugar, consideraremos qué limitaciones deseamos poner sobre los diferentes tipos de sentencias que el programa va a ser capaz de tratar. En esta versión Eliza tratará un número relativamente pequeño de formatos de sentencias. (De hecho, como vamos a finalizar con una versión acortada de Eliza, le llamaré Liz pidiendo disculpas por su naturaleza restringida.) En el fragmento de conversación que expuse anteriormente, existen dos únicos formatos:

Yo ⟨⟨verbo⟩⟩ ⟨⟨objeto⟩⟩

y

Sí

La notación que he empleado anteriormente es suficientemente explicativa por sí misma. Cuando se hace referencia a una clase de palabras, se muestra entre ángulos (⟨⟨ , ⟩⟩). Cuando está presente una palabra en particular, aparece sin enmarcar.

Ahora, si Liz encuentra "Yo ⟨⟨verbo⟩⟩ ⟨⟨objeto⟩⟩" podrá responder:

Por qué supone que Vd. ⟨⟨~verbo⟩⟩ ⟨⟨objeto⟩⟩

De nuevo, he presentado una pieza extra en la notación. El signo "~" indica que la forma verbal puede necesitar ser modificada, sobre todo si el verbo es irregular.

Si la conversación hubiera comenzado:

Persona: Me siento algo cansado hoy.

Liz podría responder:

¿Por qué supones que te sientes algo cansado hoy?

donde la única transformación es "te" por "me" y "siento" por "sientes".

Realmente voy a avanzar más rápidamente, desocupándome aquí de la gramática, para tratar las formas de gerundio (como "pensando")

como parte del objeto en lugar de como parte del verbo, ya que son tan fijos como el objeto y están obligados a precederle.

Por supuesto, si Liz realizara siempre esta transformación exactamente de esta forma, sería muy aburrido; por tanto deben existir transformaciones opcionales, seleccionadas al azar. Una de ellas se muestra en "Por tanto, tuvo Vd. una mala noche", cuyo formato general es:

Por tanto, Vd. ⟨⟨~verbo⟩⟩ ⟨⟨objeto⟩⟩

Liz tendrá dificultades con las respuestas monosílabas como "sí" porque no aportan nada a Liz para que trabaje. A este tipo de respuesta las llamaré de "no compromiso", ya que sus miembros nunca aportan nada a lo que ya se ha dicho. Algunos ejemplos son "No", "correcto" y "De acuerdo". Este último no es un monosílabo, pero todavía hay alguno peor. Dentro de la misma categoría están "Veo que" y "pensaría así", pero ¡éstas además incluyen *verbos*! de forma que Liz en estos casos generaría probablemente alguna prosa inmortal como "Por qué supone que pensaría así?

En mi diálogo de ejemplo, Eliza contendió con la respuesta de no compromiso volviendo al último objeto. El formato general es:

Cuéntame más sobre el ⟨⟨objeto⟩⟩

Bien, realmente eso no es del todo cierto. Nos llevaría a:

Cuéntame más sobre una mala noche.

que suena más pomposo. Por otra parte, intentar hacer la transformación de "una" a "su" sería contraproducente. Después de todo, es un fragmento del idioma con una particular idiosincrasia. Implica la posesión de la noche. Sería mejor solución transformar un artículo indeterminado "una" en el artículo determinado "la" en base a que todo lo que se refiera a ello ha ocurrido con anterioridad. Cuéntame más sobre la mala noche no suena demasiado mal.

Liz también se encontrará en dificultades con las preguntas, ya que no conoce ninguna respuesta. Lo más sencillo es asumir que el interlocutor humano está capacitado para introducir un signo de interrogación para indicar la presencia de una pregunta. Luego Liz puede obrar a la defensiva y ofrecer una serie de respuestas almacenadas tales como "Me temo que no lo sé" o "No puedo responderle".

Esto nos lleva a otra posible transformación del objeto (la primera fue "una" que se convirtió en "la"). Si Liz dice:

No puedo responder

invita a la respuesta:

Yo no me imagino que Vd. pueda

Liz identificará "imagino" como verbo, así pues, si utiliza el formato "por tanto Vd." "~verbo" "objeto", aparecerá con:

Por tanto tú te imaginas que tú puedes

cuando se hubiera dicho:

Por tanto tú no te imaginas que yo puedo

En otras palabras, "tú" debería haberse convertido en "yo". Ahora está claro que las transformaciones del objeto a veces son necesarias.

Otro pròblema a solucionar es el de las frases que no conforman las estructuras sintácticas discutidas hasta ahora. No parece probable que vayamos a repasar todas las posibilidades sin experimentar, por lo que el programa debería tener un mecanismo que permitiera introducir, de forma sencilla, nuevas reglas según se hicieran necesarias. Sin embargo, debido a la propia naturaleza de la conversación, muchas frases pueden tener una estructura que puede reducirse al formato "Yo ⟨⟨verbo⟩⟩ ⟨⟨objeto⟩⟩". Por ejemplo:

Me parece que el tiempo ha sido muy malo

y:

El tiempo ha sido muy malo

y ambos equivalentes, más o menos, a:

Creo que el tiempo ha sido muy malo.

Por tanto, podemos considerar la escritura de un pre-procesador que forme la última sentencia a partir de cualquiera de las dos primeras. El primer paso del algoritmo para conseguir todo esto podría ser:

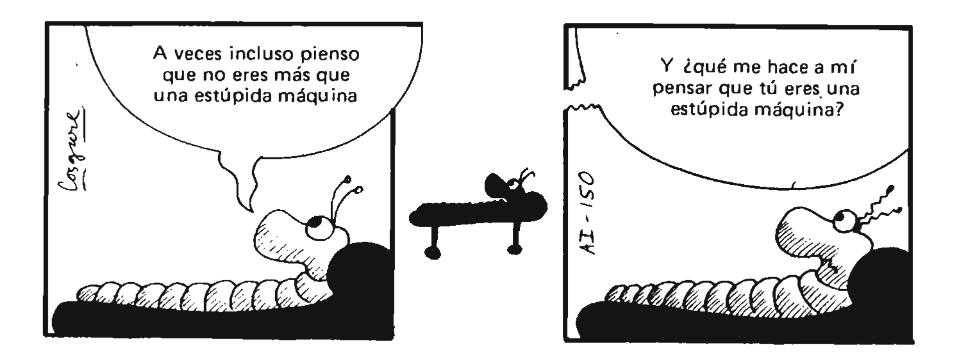
1. Investigar la frase de entrada de derecha a izquierda.
2. Si el primer símbolo es un interrogante THEN *parada*: GOTO (1).
3. Búsqueda del verbo (*buscaver*).

4. IF la palabra precedente no es "Yo" THEN añadir "Yo pienso/creo que el/la" antes de esta palabra.
5. Transformar «objeto» → «~objeto».
6. Seleccionar entre: Por tanto Vd.
Porqué piensa Vd. que
Porqué supone Vd. que
Me pregunto porqué Vd.
—«~verbo» «objeto»

Hay un par de puntos a considerar. El primero es que *buscaver* puede no ser satisfactorio al no encontrar ningún verbo, particularmente en el caso de las respuestas de no-compromiso. En tales casos encomendaremos a la operativa de búsqueda de verbo, la responsabilidad de generar una frase como "cuéntame más sobre «~objeto». Por supuesto que «~objeto» será entonces el último objeto requerido. El segundo punto, es que *buscaver* necesita reconocer un verbo cuando se lo encuentre, por lo que tendremos que facilitar un vocabulario que incluya las terminaciones irregulares. Para esto debemos considerar una estructura de datos apropiada. He aquí una posibilidad muy sencilla:



Cada entrada de v$ contiene una forma de cada verbo sin ninguna ordenación. Ahora *buscaver* puede realizar una búsqueda lineal en esta lista.



EL CODIGO

Intentemos ahora codificar algo de esto. Sabemos algunas de las rutinas que necesitamos, por lo que podemos asignarlas algunas líneas de comienzo de su codificación:

```
10  LET parada = 1000:LET buscaver = 1200:
    LET transobj = 1400
```

Necesitamos algunos arrays, de los cuales sólo de uno conocemos lo suficiente para dimensionarlo:

```
20  DIM v$(100,10): LET o$ = "  "
```

Esto nos permitirá trabajar con cien verbos y un máximo de diez letras por verbo. Habrá observado que también he inicializado el objeto, o$, a espacios. Así evitaremos que Liz intente manipular un objeto inexistente. Ahora solicitaremos la primera entrada del usuario, tomamos la primera frase y continuamos con el algoritmo:

```
100   PRINT "Hola, ¿qué quieres decirme?
110   INPUT s$
115   PRINT INK4; s$
120   IF s$(LEN s$) = "?" THEN GOSUB parada: GOTO 110
130   GOSUB buscaver
```

Necesitamos definir exactamente lo que devuelve *buscaver*. Por el momento supondremos que es muy sencillo y que da un apuntador, p, a la primera letra de un verbo identificado:

```
140   LET o$ = s$(p TO)
```

```
150    IF s$(p – 2) <> "I" THEN LET ep = p – 2:
       GOSUB buscapal:LET o$ = "pienso que el" + w$ + " " + o$
```

Observe que aquí se supone que existe exactamente un espacio entre cada palabra. Suponer esto es algo natural, y en cualquier caso, sería fácil escribir un fragmento de codificación que agrupara el texto de entrada, eliminando los espacios extra y las contracciones si las hubiera, que hasta ahora no las estamos permitiendo. Acabo de presentar una nueva subrutina llamada *buscapal.* Su función es la de devolver en w$ la palabra anterior a la apuntada por p. Se pasa ep, que apunta al final de la palabra a buscar:

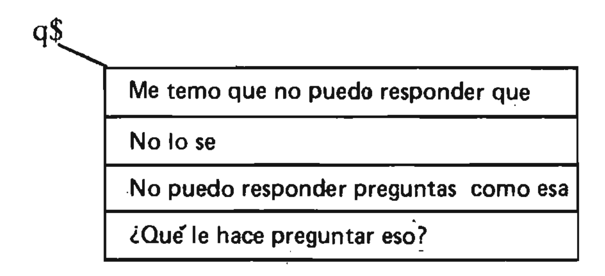
```
160    GOSUB transobj
170    GOSUB select
180    PRINT r$ + o$
190    GOTO 110
```

transobj transforma el objeto y el verbo (en realidad es el predicado) según sea necesario y devuelve el resultado en o$. La subrutina *select* genera una selección al azar a partir de las frases de comienzo posibles de r$.

Ahora podemos comenzar a examinar en detalle las subrutinas.

parada (1000)

Esta subrutina es sencilla. Todo lo que necesitamos hacer es seleccionar de un conjunto de respuestas estándar, organizadas como, por ejemplo, éstas:



Así la línea 20 se convierte en:

```
20  DIM v$ (100,10):DIM q$(4,40)
```

La codificación es:

```
1000  LET r = INT(RND * 4) + 1
1010  PRINT q$(r)
1020  RETURN
```

No estoy desarrollando codificación para establecer los arrays de texto como q$ y v$ porque existen diferentes formas en que podría Vd. hacerlo, todas ellas eficaces. Por ejemplo, puede utilizar sentencias READ y DATA o puede inicializarlos en modo inmediato. Es conveniente hacer esto último, ya que entonces no existe evidencia de las palabras clave o las frases del listado; así la operativa del programa no es transparente.

buscaver (1200)

```
1200  LET ep = LEN s$
1210  GOSUB buscapal
1220  LET t$ = w$
1230  FOR v = 1 TO 100
1240  IF t$ = v$(v) THEN RETURN
1270  NEXT v
1280  LET ep = p − 2
1290  IF p > 1 THEN GOTO 1210
1300  RETURN
```

Necesita una pequeña explicación. A *buscapal* se le da primeramente un apuntador a la última palabra de s$ y la rutina la devuelve en w$, junto con p, que apunta a su comienzo. Como todos los verbos en v$ tienen diez bytes de longitud, tenemos que compararlo con un campo de 10 bytes, así pues lo copiamos en t$ que dimensionamos (DIM) con diez de longitud. Ahora lo comparamos con cada verbo de v$. Si no se encuentra coincidencia, ep se disminuye en 2 para que apunte al final de la palabra anterior. El proceso continúa hasta que p = 1 cuando se ha alcanzado el comienzo de s$ sin encontrar ningún verbo conocido.

Observe que si *buscaver* devuelve p = 1, es que no se encuentra ningún verbo reconocible, por lo que se invocaría a una rutina llamada *barquillo* ya que Liz no está seguro de lo que está pasando. Esto nos sugiere una línea de codificación adicional en la rutina principal:

```
135   IF p = 1 THEN GOSUB barquillo:GOTO 110
```

Cuando esté Vd. probando esto, es posible que existan algunas llamadas a *barquillo* no esperadas debido a que se haya olvidado de introducir algún verbo obvio en v$, por lo cual necesitará en realidad una rutina adecuada para añadir entradas a v$.

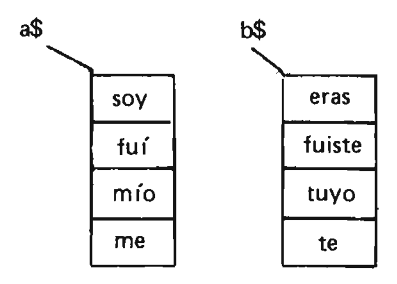
transobj (1400)

Existen hasta ahora tres transformaciones que hemos considerado necesarias:

1. Terminaciones verbales entre la primera y segunda persona.
2. Artículos indefinidos como "un" "una" "algunos/as" que se convierten en "el" "la" "los" "las".
3. Pronombres en primera persona que se transforman en segunda persona y viceversa. Así "me" se convierte en "te"; "mío" en "tuyo".

Examinaremos o$ de derecha a izquierda nuevamente, buscando estas palabras clave y construyendo el objeto transformado en m$. La transformación tipo 2 es un caso especial en cuanto a que la transformación se efectúa en un solo sentido, es decir, "un" se convierte en "el" pero no a la inversa; por tanto la trataremos por separado. Por otro lado "soy" se transforma en "eres" y viceversa; "mío" en "tuyo" y viceversa y así sucesivamente.

Por lo tanto, podríamos disponer de dos arrays organizados de la siguiente forma:



De nuevo, constan de diez bytes para mayor coherencia.

Si se encuentra una palabra en a$, la sustituimos por su equivalente de b$ y viceversa. Así pues, la codificación es:

```
1400  LET ep = LEN o$
1410  LET s$ = o$:LET o$ = " "
```

```
1420  GOSUB buscapal
1425  LET t$ = w$
1430  IF w$ = "un" OR w$ = "una" or w$ = "algunos/as"
      THEN LET w$ = "the":GOTO 1480
1440  FOR a = 1TO 4
1450  IF t$ = a$(a) THEN LET w$ = b$(a):GOTO 1480
1460  IF t$ = b$(a) THEN LET w$ = a$(a):GOTO 1480
1470  NEXT a
1480  LET o$ = w$ + o$
1490  IF p> 1 THEN LET ep = p – 1:GOTO 1420
1500  RETURN
```

Observe que primeramente se transfiere o$ a s$ para que *buscapal* pueda utilizarla. Luego, el objeto transformado se puede construir en o$. No se olvide tampoco de restaurar ep (línea 1490) antes de llamar de nuevo a *buscapal.*

buscapal (1600)

No existe ningún problema en esta rutina.

```
1600  LET p = ep – 1
1610  IF p>0 THEN IF s$(p) <> " " THEN LET p = p – 1:
      GOTO 1610
1620  LET p = p + 1
1630  LET w$ = s$(p TO ep)
1640  RETURN
```

Observe que la línea 1610 no podría ser:

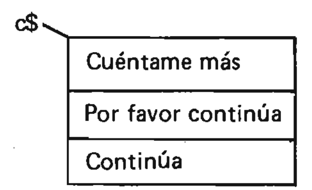
```
1610  IF p>0 AND s$(p) <> " " ...
```

ya que si p es cero, BASIC trataría de comprobar s$(p) y Vd. obtendría un error de índices.

barquillo (1800)

La rutina *barquillo* puede hacer dos cosas básicas. Puede ser algo no comprometedor, como decir "continúe" o puede utilizar el objeto anterior (que estará todavía en o$) para formar una frase como "Cuéntame más sobre tus padres".

Necesitamos otro array más:



Estas tres frases se han seleccionado cuidadosamente de forma que cada una de ellas se pueda utilizar por sí misma o seguidas de "sobre" y luego el contenido de o$. Bueno, aproximadamente o$. Ya no necesitamos más el verbo. Por tanto necesitamos un apuntador al primer espacio de o$.

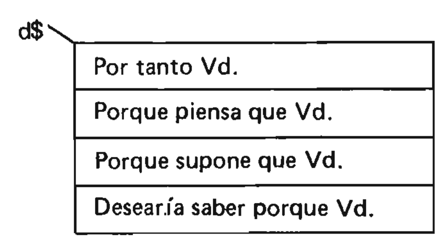
```
1800  FOR p = 1 to 10
1810  IF o$(p) = " " THEN GOTO 1830
1820  NEXT p
1830  LET r = INT(RND * 3) + 1
1840  LET r$ = c$(r)
1850  IF RND > 0.5 THEN LET r$ = r$ + "sobre porqué" + o$
1860  PRINT r$
1870  RETURN
```

La línea 1820 determina si la frase seleccionada de c$ se deja tal como está o si se incorpora a ella el objeto anterior. Cualquiera de las formas es igualmente probable, pero, obviamente Vd. puede jugar con el 0,5 a fin de alterar las probabilidades de acuerdo con sus gustos personales.

select (2000)

```
2000  LET r = INT(RND * 4) + 1
2010  LET r$ = d$(r)
2020  RETURN
```

Aquí se asume que d$ se ha creado con:



Cuentos

Hasta ahora he mantenido deliberadamente las características de estas rutinas al mínimo, por dos razones. La primera de ellas que no deseaba que los árboles no dejaran ver el bosque y la segunda que Liz consume gran tiempo en búsquedas, que son bastante lentas en BASIC, por lo cual añadir complicaciones la harían incluso más lenta.

Así pues, lo primero que observará cuando ejecute Liz será una gran cantidad de espacios producidos por la impresión de líneas de longitud fija de los arrays de cadenas bidimensionales. Por ejemplo, si d$ tiene 30 bytes y *select* selecciona la opción "Por tanto Vd.", obtendrá 23 espacios extraños entre "Por tanto Vd." y el objeto.

Sería útil disponer de una rutina que los eliminara.



Triturador (2200)

La rutina *triturador* aceptará cualquier cadena en k$ y la copiará en l$ ignorando los espacios sobrantes. Se podría utilizar también en la entrada para anular los posibles espacios innecesarios que pudiese introducir el usuario.

```
2200  LET l$ = " "
2210  FOR p = 1 TO LEN k$
2220  IF k$(p) <> " " THEN LET l$ = l$ + k$(p):LET espacio = 0
2230  IF k$(p) = " " THEN LET espacio = espacio + 1
2240  IF espacio = 1 THEN LET l$ = l$ + " "
2250  NEXT p
2260  RETURN
```

Observe que *triturador* asume que el primer carácter en k$ no es un espacio.

Prueba

Inténtelo. La primera conversación (poco menos que sin sentido) que mantuve con Liz, fue algo así:

Liz: Hola, ¿qué quieres contarme?
Yo: Hoy me siento algo irritado.
Liz: Desearía saber por qué te sientes hoy algo irritado.
Yo: La mafia me vigila.
Liz: ¿Por qué supones que la mafia te vigila?
Yo: Se esconden tras las esquinas y me espían.
Liz: Cuéntame más sobre por qué piensas que la mafia te vigila.
Yo: ¿Qué desearías saber?
Liz: Me temo que no puedo contestar a eso.
Yo: OK.
Liz: Por favor continúa.
Yo: Ellos tienen mucho cuidado de que no les vea.
Liz: ¿Por qué piensas que el ellos tienen mucho cuidado de que no les vea (1)?
Yo: Pienso que ellos son muy listos.
Liz: Desearía saber por qué piensas que el ellos son muy listos (2).
Yo: Tú no me crees.
Liz: ¿Por qué supones que el yo no te creo? (3)

Bien, ya lo advertí al comienzo que esto era un primer intento del algoritmo. Veamos qué modificaciones sugieren las formas no gramaticales señaladas (1), (2) y (3).

Todas ellas tienen una cosa en común: aparecen otros pronombres distintos a "yo". Liz los ha tratado como si fueran *sustantivos* y pone artículos determinados delante de ellos.

Un segundo problema es que se han efectuado transformaciones incorrectas de los verbos, ya que la segunda persona del singular y todas las formas plurales son indistinguibles. Así pues, mientras la conversación transcurra entre tú y yo, "soy" y "eres" se intercambian satisfactoriamente, pero cuando aparece "ellos", Liz se hace un pequeño lío ya que "son" no se puede cambiar por "soy".

Finalmente, la transformación me-te funciona bien, pero la inversa no está garantizada ya que el pronombre de la primera persona debería ser "yo" si es el sujeto de un párrafo.

Checksub (2400)

Escribiremos una rutina *checksub* para examinar estos casos especiales. La línea 150 queda así:

```
150  GOSUB checksub
```

y *checksub* será así:

```
2400  LET vt = 1
2410  IF s$(p – 2) = "Yo" THEN RETURN
2420  LET ep = p – 2: GOSUB buscapal
2430  IF w$ = "Vd." THEN LET o$ = "pienso que yo" + o$:RETURN
2440  IF w$ = "nosotros" OR w$ = "ellos" OR w$ = "ello"
      THEN LET o$ = "piensan/mos" + w$ + " " + o$:
      LET vt = 0:RETURN
2450  LET o$ = "piensa que él" + w$ + " " + o$:
      LET vt = 0:RETURN
```

Esta rutina tratará bien los problemas derivados de los pronombres y hace algo más, devuelve vt = 1 si es necesaria una transformación de verbo y vt = 0 en caso contrario. Necesitamos modificar *transobj* para tener esto en cuenta. ¡Es fácil!. Inserte:

```
1405  LET st = 3
1406  IF vt THEN LET st = 1
```

y edite la línea 1440:

```
1440  FOR a = st TO 4
```

Ahora, si se requiere una transformación de verbo, st es 1 como antes, pero en caso contrario el bucle comienza en 3, evitando así conjuntamente las parejas soy-eres y fui-fuiste.

Algunas reflexiones finales

Liz no es perfecto y nunca lo podrá ser. Si yo digo:

El tiempo es algo desapacible

Liz puede responder

Así pues Vd. piensa que el tiempo es el algo desapacible.

¡Imagínese en una salida de este tipo!

Todavía existe una amplia gama de ajustes que se pueden hacer para mejorar su ejecutoria. Incremente el rango de frases que utiliza Liz; sustituya "imaginar" o "creer" por "pensar" al azar; incremente su vocabulario de verbos.

He aquí algunas sugerencias más ambiciosas para agilizar sus músculos mentales.

Proyectos

1. Modificar *barquillo* para que la conversación pueda volver a uno de los temas anteriores, no al más reciente. Esto asegurará la creación de un array para o$ y la selección de uno de sus elementos al azar. Comoquiera que este array no puede ser indefinidamente largo, en algún punto se deberán descartar los antiguos objetos en favor de los más recientes. Sin embargo, no olvide que en la conversación humana, no ocurre con frecuencia esta especie de marcha atrás; por tanto debería haber un cierto peso a favor del tema más reciente.
2. Por ahora, el vocabulario de Liz es fijo. Sería posible permitirle que, hasta cierto punto, aprendiera verbos de su interlocutor humano. La palabra que sigue a un "yo" es muy probable que sea un verbo y con cierta menos probabilidad la que sigue a un "tú", "él", "nosotros", etc., también lo sea. Así pues Liz podría preguntar si una palabra es un verbo cuando la encuentra desconocida detrás de un pronombre y añadirla a v$ si obtiene una respuesta afirmativa.

   Sin embargo, tenga cuidado con esto. Recuerde un consejo de alguien que dijo:

   "Casi todos los sustantivos pueden ser formas verbales".

3. Unas veces Liz le hace una pregunta y otras forma una afirmación. Por el momento, en ninguno de ambos casos inserta el signo de interrogación. Modifíquelo para que lo inserte de forma apropiada.
4. Permita que Liz trate ciertas contracciones como a el (al) de el (del) etc. Haga también que Liz puede tratar correctamente las letras mayúsculas (por ahora, no transformará "Tú" en "me", por ejemplo).

8
Representación del conocimiento

Hemos visto lo fácil que es engañar a un intérprete de lenguaje natural que dispone solamente de reglas de sintaxis (¿recuerda el maíz y el elefante?) y cómo se enfrenta Liz con los problemas, ignorándolos. Por supuesto, no podemos permitirnos una solución tan fácil; no sería de utilidad.

Así pues, tenemos el problema de cómo vamos a dar a la máquina un "modelo universal"; de cómo, en realidad, podemos representar en ella el *conocimiento*.

Para percatarnos de la dificultad que encierra este problema y cómo lo resolvemos los humanos, aparentemente sin esfuerzo, consideremos el párrafo siguiente:

> Julia pensó que podría conducir hasta la casa de Ana. Tocó el timbre de la puerta, pero no obtuvo respuesta. Así pues, volvió a la ciudad.

Consideremos ahora cuánto debe conocer un sistema para responder preguntas como:

1. ¿Condujo Julia hasta la casa de Ana?
2. ¿Quién tocó el timbre de la puerta?
3. ¿Estaba Ana en casa?
4. ¿Quién volvió a la ciudad?

En lo que se refiere a la pregunta 1, sabemos que Julia actuó conforme a su pensamiento, aunque no se diga. De otro modo no hubiera podido tocar el timbre de la puerta, deducción que implica que conozca-

mos cosas tales como la situación de los timbres. La pregunta 2 implica la misma información y también necesita conocer que no es probable que Ana tocase su propio timbre de la puerta, a menos, por supuesto, que lo esté probando; pero no hay nada que haga suponer esto. También deducimos que Ana no estaba en casa, ya que cuando la gente está en casa responde al timbre. Sabemos también que es Julia quien volvió a la ciudad, ya que el paradero de Ana es desconocido.

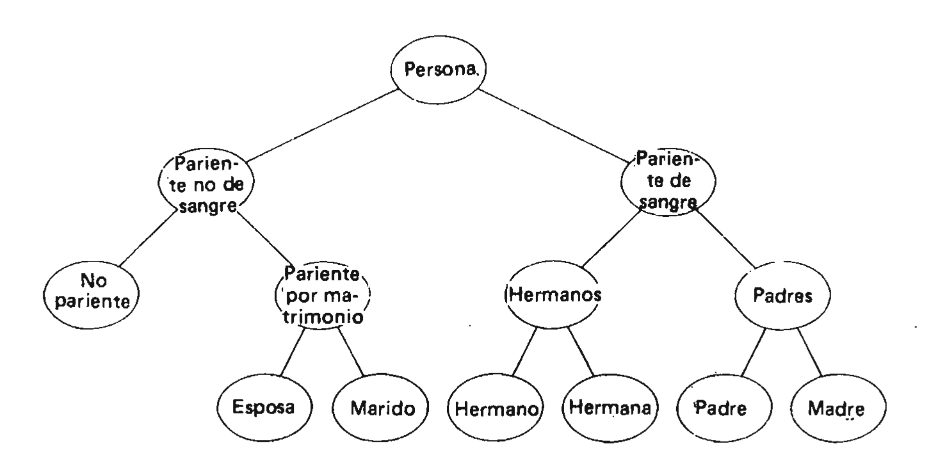
Así es como interpretaría el párrafo un ser humano, basado en su conocimiento (o el sentido común si lo prefiere). Pero existe una alternativa coherente con los hechos conocidos. Es que Julia cambiara de idea sobre visitar a Ana, tocara su propio timbre, viera que estaba estropeado y fuera a la ciudad, (¡quizás para localizar a un electricista!).

Hasta aquí, en lo que se refiere al ser humano, existe el vacío sutil de una secuencia apropiada sobre esta interpretación. Hacer que una máquina presente este grado de sutileza es una torpeza mental.

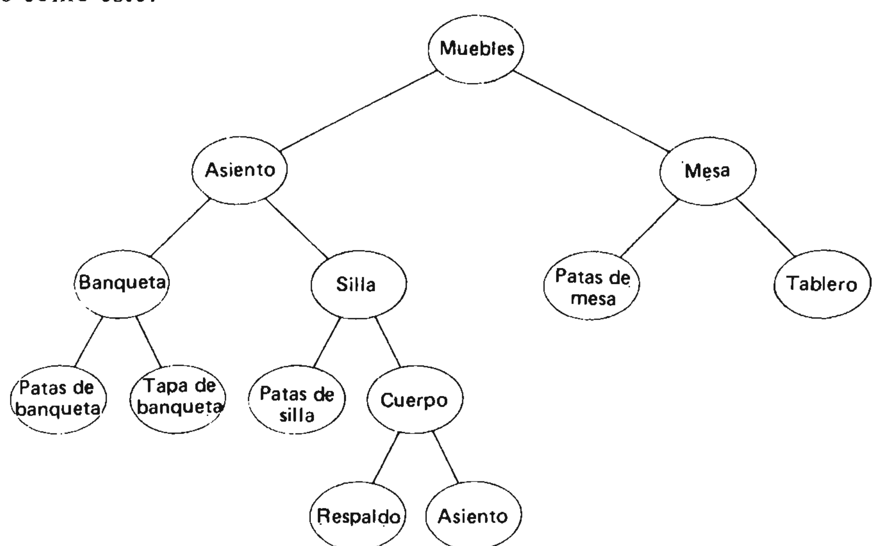
Sin embargo, podemos describir unas formas de conexión más simples de forma relativamente fácil.

ARBOLES DE CONOCIMIENTOS

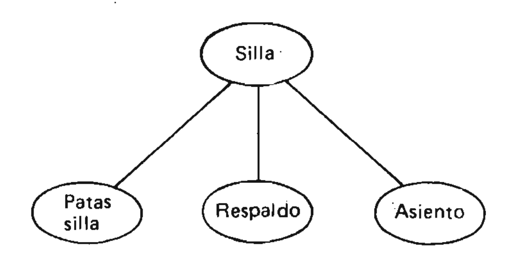
Supongamos que dice Vd. a Liz, "Yo tuve una discusión con mi padre". Sería bueno que Liz pudiera decir "Cuéntame más sobre tus padres". Para ello Liz necesitaría saber que existe una relación entre "padre" y "padres" (en el sentido de padre o madre). Podemos formalizar estas relaciones por medio de árboles como éste:



o como éste:



Existen varios puntos a considerar sobre estos dos ejemplos. Lo primero es que están incompletos. No se menciona a los abuelos en el primer caso o a los bancos del parque en el segundo o cualquier otra de las muchas posibilidades que existen. En segundo lugar, he organizado los árboles de forma que de cada nodo del árbol no salen más de dos ramas. Podría haber escrito, por ejemplo:



pero deliberadamente he inventado el término "cuerpo" para evitar más ramas. "Cuerpo" es una especie de nodo falso, ya que no es un término que utilicemos comúnmente en este contexto. Todo esto se reduce a que siempre es posible formar un árbol con dos ramas a lo sumo partiendo de cada nodo (árbol binario) introduciendo ese tipo de nodos falsos. Este tipo de escamoteo nos ayudará más tarde a resolver los problemas de la codificación.

Otro punto adicional; el mismo término puede aparecer en varios sitios del árbol ("patas" aparece en tres sitios del árbol de muebles, por ejemplo). En un sistema más sofisticado quizás se prefiera disponer sólo de un nodo llamado "patas" y varias formas partiendo de él. Esto nuevamente dificultaría las cosas más de lo deseable en este momento, por lo que es mejor distinguirlas como "patas de silla", "patas de mesa" y así sucesivamente.



LA ESTRUCTURA INTERNA DE LOS DATOS

Cada nodo puede quedar representado por un array tipo cadena que alberga su contenido más dos apuntadores a sus nodos hijos. Así, el árbol de los muebles aparecería así:

	n$	p	
1	muebles	2	3
2	asiento	4	5
3	mesa	6	13
4	banqueta	12	7
5	silla	8	9
6	patas mesa	Ø	Ø
7	tapa banqueta	Ø	Ø
8	patas silla	Ø	Ø
9	cuerpo	1Ø	11
1Ø	respaldo	Ø	Ø
11	asiento	Ø	Ø
12	patas banqueta	Ø	Ø
13	tablero	Ø	Ø

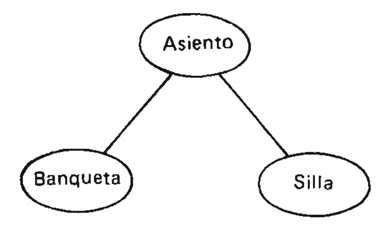
Como hemos prometido utilizar árboles binarios, p tiene solamente dos columnas. Observe que el orden de los nodos en n$ carece de importancia. Los apuntadores cambiarán simplemente para adecuarse. Utilizo ceros en p para indicar que se ha alcanzado un nodo terminal (hoja).

CRECIMIENTO DE ARBOL

Deseamos poder facilitar al programa información sobre los muebles (o sobre personas o lo que sea) en cualquier orden, tal como se nos antoje. Por tando deberíamos ser capaces de algo como:

Las banquetas y las sillas son miembros del tipo asiento

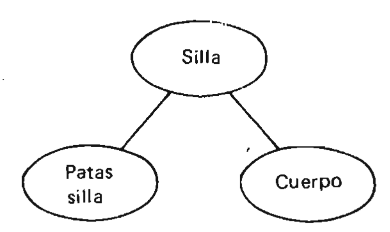
y esperar que el sistema genere:



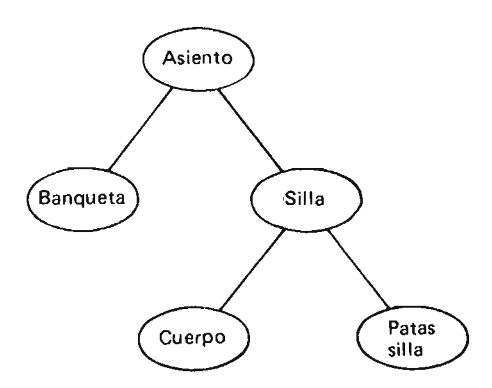
o:

Las patas y el cuerpo son parte de la silla

y obtener:



Por supuesto que si estas dos operaciones se ejecutan en este orden, el programa deberá ser capaz de percatarse de que "silla" se ha mencionado anteriormente y hacer crecer al árbol en sentido descendente para formar:



De esta forma, se pueden añadir nuestros conocimientos básicos de forma absolutamente natural. Después de todo, en el mundo real, el conocimiento es fruto de la experiencia y las experiencias nos abordan de forma totalmente impredecible.

Supongamos que asumimos que nunca vamos a tener más de 100 nodos en nuestro árbol. Podríamos comenzar con:

```
10  DIM n$(100,20):DIM p(100,2)
20  LET crecer = 1000:LET buscar = 2000:LET pff = 1
30  INPUT "crecer o buscar (g/s)"; s$
40  IF s$ = "g" THEN GOSUB crecer
50  IF s$ = "s" THEN GOSUB buscar
60  GOTO 30
```

de forma que podemos añadir información al árbol con *crecer* o averiguar algo que ya conoce con *buscar*. Observe la inicialización de pff. Es el Apuntador hacia la Primera posición Libre del árbol.

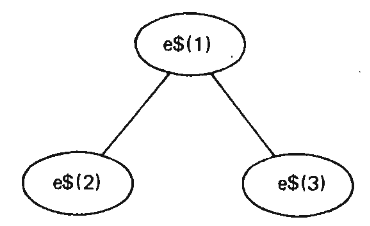
Crecer (1000)

La primera tarea será extraer los datos a añadir. Algo como:

```
1000  INPUT "Partes de"; e$(1); "son"; e$(2); "y"; e$(3)
```

lo hará, al menos para el ejemplo de muebles. El término "Partes de" no siempre será significativo ("Partes de padres son madre y padre" no tiene mucho sentido, por ejemplo), así pues lo recordaremos para modificarlo más adelante.

Tal como están las cosas, el problema consiste en formar el subárbol:



y una frase típica de entrada sería:

Partes de la banqueta son las patas de banqueta y la tapa de banqueta

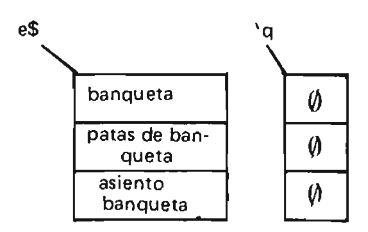
en cuyo caso e$(1) es "banqueta", e$(2) es "patas de banqueta" y e$(3) es "tapa de banqueta". El array de entrada e$ se dimensionará:

```
DIM e$ (3,20)
```

para que coincida con n$. También mantendremos en un array, q un registro que contenga dónde apuntan los elementos de e$ (si ha lugar):

```
DIM q (3)
```

de forma que tendríamos:



para comenzar. Ahora buscaremos en n$ cada elemento de e$ para emparejarlos. Si encuentra uno, hace que el elemento correspondiente de q apunte adonde se le encontró:

```
1010  FOR i = 1 TO 3
1020  LET q(i) = 0
1030  FOR r = 1 TO 100
1040  IF n$(r) = e$(i) THEN LET q(i) = r:GOTO 1060
```

```
1050  NEXT r
1060  NEXT i
```

Si esta rutina de búsqueda no encuentra emparejamiento, el elemento correspondiente de q permanece a cero; así sabemos que se debe añadir a n$.

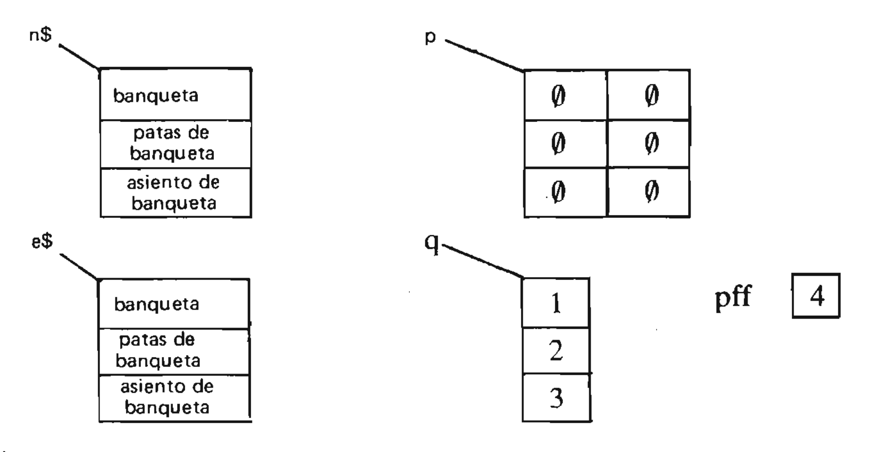
```
1070  FOR i = 1 TO 3
1080  IF q(i) <> 0 THEN GOTO 1120
1090  LET n$(pff) = e$(i)
1100  LET q(i) = pff
1110  LET pff = pff + 1
1120  NEXT i
```

Cuando identificamos un elemento a añadir a n$, se le incorpora en la fila pff (línea 1090). Luego, se registra su posición en q, y se incrementa pff. Ahora todo lo que se necesita es transferir esos enlaces a p:

```
1130  LET p(q(1),1) = q(2)
1140  LET p(q(1),2) = q(3)
1150  RETURN
```

Para aclarar cómo actúa esta rutina, vamos a seguirla para las tres primeras entradas (banqueta, patas de banqueta y tapa de banqueta) y luego añadiremos "Partes de un asiento son banqueta y silla".

Por supuesto, es posible que no se encuentre emparejamiento en n$ para las tres primeras entradas; si fuera así, las líneas 1010 a 1060 ejecutarían una búsqueda infructuosa sin tomar acción alguna. En consecuencia, en el bloque de codificación siguiente (líneas 1070 a 1120) las tres entradas de e$ se añaden a n$ (q(i) siempre es cero) y tenemos:



Examinemos ahora el efecto de las dos últimas líneas de la rutina; q(1) = 1 y q(2) = 2; por tanto, la línea 1130 se convierte en realidad en:

```
LET p (1,1) = 2
```

y de forma semejante, la línea 1140 en:

```
LET p (1,2) = 3
```

los ceros de la fila 1 de p se sustituyen por 2 y 3, apuntando correctamente a "patas de banqueta" y "tapa de banqueta". Cuando añadimos asiento, banqueta y silla tenemos sobre la entrada:



(En realidad esto no es cierto del todo, ya que q no vuelve a valer cero hasta que entra la rutina de búsqueda, pero el efecto es el mismo).

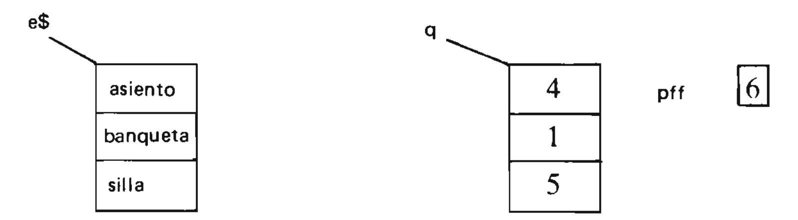
Esta vez la búsqueda detecta una referencia previa, por lo que se obtiene:



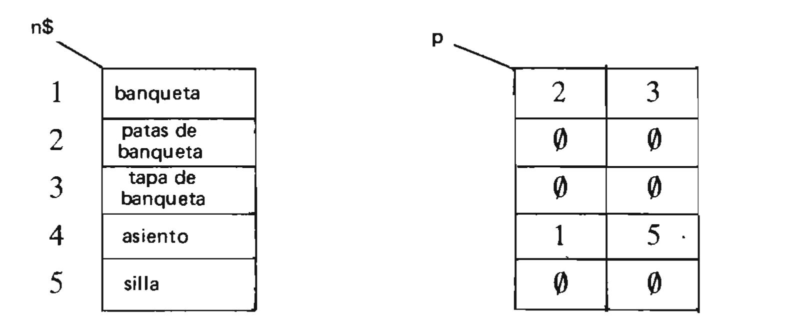
ya que encuentra "banqueta" en la fila 1 de n$ y almacena esta información en q; n$ tiene ahora "asiento" y "silla" incorporados y sus números de fila son devueltos a q, obteniendo:

n$
banqueta
patas de banqueta
tapa de banqueta
asiento
silla

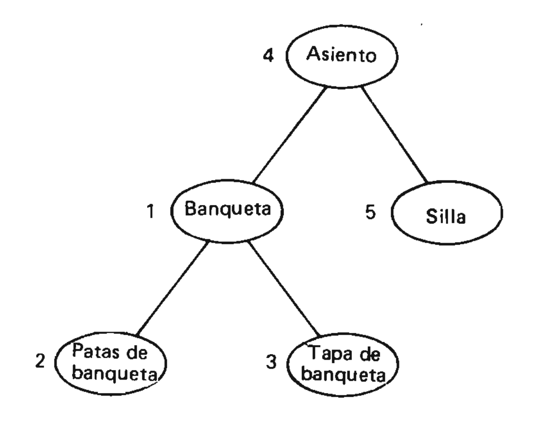
p	
2	3
∅	∅
∅	∅
∅	∅
∅	∅



Luego p (4,1) toma el 1 y p (4,2) el 5:



que da lugar al árbol:



RECORDANDO CONOCIMIENTOS

Ahora tenemos que escribir *búsqueda*, pero antes de ello, deberíamos considerar qué tipos de operaciones de búsqueda necesitamos realizar sobre el árbol.

Por ejemplo, quizás deseemos conocer el nombre genérico para un elemento ya mencionado. En nuestro ejemplo, será siempre "muebles" o en general la raíz del árbol. Aquí debemos tener cuidado. Algunas ramas del árbol son más precisas que otras. Si Liz utilizara un árbol como éste y Vd. dijera "Yo rompí una pata de una silla" podría responder "Cuéntame más sobre el mueble" que es un tipo de respuesta improbable en un ser humano. Volveremos más tarde a este problema.

Es más probable que deseemos saber algo sobre los nodos próximos a algún nodo en particular. Existen dos posibilidades obvias:

1. Si no es un nodo raíz, ¿cuál es su madre?
2. Si no es una hoja, ¿cuáles son sus hijas?

Finalmente, podríamos preguntar qué conexión existe entre dos nodos dados. Esto implica medir el grado de conexión, el cual debería definirse. La medida más simple consiste en contar el número de ramas entre los dos nodos. Por ejemplo, la distancia entre "patas de mesa" y "respaldo" es de 6, mientras que la distancia entre "silla" y "tapa de silla" es sólo de 2. También puede ser útil observar si dos nodos están en la misma rama. Por ejemplo, las distancias entre "banqueta" y "patas de silla" y entre "asiento" y "respaldo" son 3 en ambos casos, pero "respaldo" es una parte (remota) de "asiento", conexión que no es verdadera para los otros dos nodos. Con esto no quiero indicar que estas medidas sean perfectas (en cierta manera están distorsionadas por la necesidad de presentar nodos falsos para mantener la naturaleza binaria del árbol), pero ciertamente dan una idea aproximada y rápida de algunas relaciones.

Así pues, *búsqueda* tiene la función básica de llamar a una de estas tres rutinas:

```
2000  INPUT "encontrar madre, hijas, conexión,
      salida (m/d/c/x)"; q$
2010  IF q$ = "m" THEN GOSUB madre:GOTO 2000
2020  IF q$ = "d" THEN GOSUB hijas:GOTO 2000
2030  IF q$ = "c" THEN GOSUB conex:GOTO 2000
2040  IF q$ = "x" THEN RETURN
2050  GOTO 2000
```

Alternativamente, Vd. podría pedir que el usuario tecleara el nombre completo de la rutina requerida y luego llamarla con:

```
2010  GOSUB VAL q$
```

pero entonces es necesario comprobar que q$ contiene una palabra definida.

madre (2200)

En primer lugar, averigüemos de quién vamos a encontrar la madre:

```
2200  DIM d$(20)
2210  INPUT "Encontrar madre de"; d$
```

y luego buscarla en n$

```
2220  FOR i = 1 TO 100
2230  IF n$(i) = d$ THEN GOTO 2250
2240  NEXT i
```

Supongamos que estamos examinando "silla", cuya madre es "asiento" y que el árbol completo se ha creado tal y como se muestra al comienzo de este capítulo. Al salir del bucle, i será 5. Ahora, por supuesto, "asiento" tiene un apuntador 5 asociado, en el array p. Por tanto, el problema ahora consiste en buscar p para el valor actual de i. La fila que lo contiene, también contiene el nodo madre apropiado en n$.

```
2250  FOR r = 1 TO 100
2260  IF p(r,1) = i OR p(r,2) = i THEN PRINT n$(r):RETURN
2270  NEXT r
```

Si recorremos el bucle sin encontrar el apuntador requerido, entonces el nodo no tiene madre:

```
2280  PRINT "El nodo es la raíz"
2290  RETURN
```

hijas (2400)

```
2400  DIM m$(20)
2410  INPUT "Averiguar las hijas de"; m$
2420  FOR i = 1 TO 100
2430  IF n$(i) = m$ THEN GOTO 2450
2440  NEXT i
```

¿No le sugiere esto nada? Pues bien, si desde las subrutinas *madre* e *hijas* hubiera llamado a otra denominada *buscanodo*, me hubiera ahorrado tener que escribir dos veces esta codificación, que es la misma que *madre*.

Sin embargo, el siguiente fragmento es diferente y muy sencillo. En el supuesto caso de que no sean cero, los apuntadores de la fila que hemos encontrado nos dan las filas para las hijas:

```
2450  IF p(i,1) = 0 AND p(i,2) = 0 THEN PRINT
      "Esto es una hoja":RETURN
2460  IF p(i,1) > 0 THEN PRINT n$ (p(i,1))
2470  IF p(i,2) > 0 THEN PRINT n$ (p(i,2))
2480  RETURN
```

conex (2600)

Esta vez comenzaremos por buscar dos nodos. Vamos pues a escribir *buscanodo*. Definamos su función. Acepta t$ (previamente dimensionado con longitud 20), busca coincidencia sobre n$ y devuelve i, que es la fila donde encuentra la coincidencia. Si no se encontrara coincidencia alguna, devuelve una variable coinc = 0; si la encuentra la devuelve a 1.

Así pues, buscanodo es:

```
2800  LET coinc = 1
2810  FOR i = 1 TO 100
2820  IF n$(i) = t$ THEN RETURN
2830  NEXT i
2840  LET coinc = 0
2850  RETURN
```

y las líneas 2200 a 2240 pueden sustituirse por:

```
2200  INPUT "Averiguar madre de"; d$
2210  LET t$ = d$:GOSUB buscanodo
2220  IF NOT coinc THEN PRINT "Nodo no encontrado":
      RETURN
```

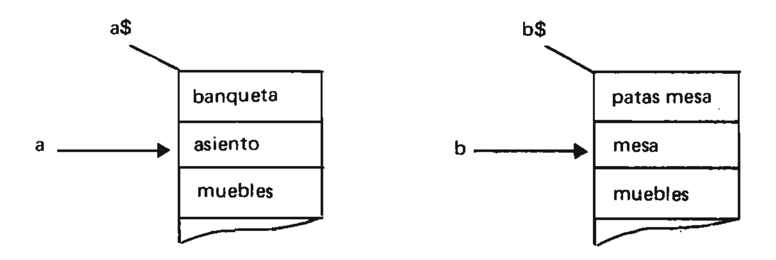
Se podría hacer un cambio semejante a *hijas* teniendo además la ventaja de poder detectar nodos desconocidos, comprobando el valor de *coinc*. Pero, volvamos al problema...

```
2600  INPUT "1er nodo"; t$
2610  GOSUB buscanodo
2620  IF NOT coinc THEN PRINT "Nodo no encontrado":
      RETURN
2630  LET j = i
2640  INPUT "2o nodo"; t$
2650  GOSUB buscanodo
2660  IF NOT coinc THEN PRINT "Nodo no encontrado":
      RETURN
2670  LET k = i
```

Así pues, ahora tenemos apuntadores a los dos nodos que estamos investigando en j y en k. ¿Qué hacer con ellos? Como de costumbre, tomemos un ejemplo para clarificar las ideas. Supongamos que creamos dos arrays tipo cadena cuyos primeros miembros sean los dos nodos recién encontrados. Si, por ejemplo, estamos conectando "banqueta" y "patas de mesa":

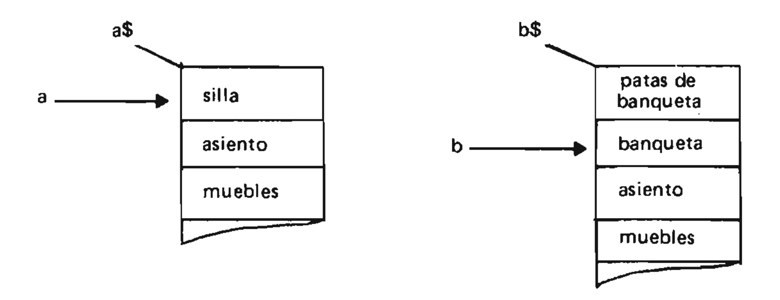


Para cada uno de ellos trazamos un camino hacia la raíz:



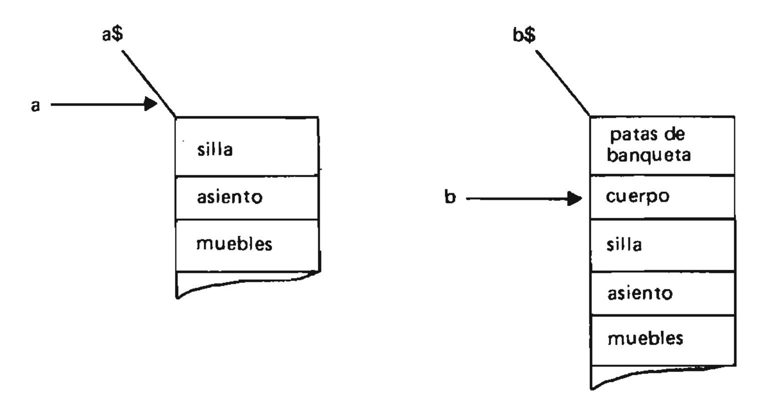
Ahora comparamos los últimos elementos incorporados a los dos arrays. Si son el mismo (como es el caso aquí), movemos ambos apuntadores un lugar hacia atrás y repetimos el proceso hasta que no sean el mismo. En este caso, los dos apuntadores (llamados a y b) se detendrán en 2. La distancia entre "banqueta" y "patas de mesa" es 4, que es igual a a + b.

Pero en este ejemplo existe simetría que generalmente no está presente. Pruebe con "silla" y "patas de banqueta":



a y b quedarán como se muestra y a + b = 3, como debe ser.

Pero, ¿qué ocurre en el caso especial de que los dos nodos estén en la misma línea, como ocurre con "silla" y "tapa de silla"?



Esta vez, uno de los apuntadores se transforma en cero antes de que los elementos correspondientes sean distintos. Así, si a o b es cero, los nodos están en línea, aun cuando sigue siendo verdadero que a + b es la distancia entre ellos (en este caso 2).

Ahora algo más de codificación. La creación de a$ y b$ es en realidad el mismo proceso, por tanto, tiene más sentido utilizar un único array y pasar los parámetros a una rutina llamada *rastreo* que determinará qué parte del array se va a examinar. La subrutina *rastreo* necesita también el apuntador al nodo en cuestión:

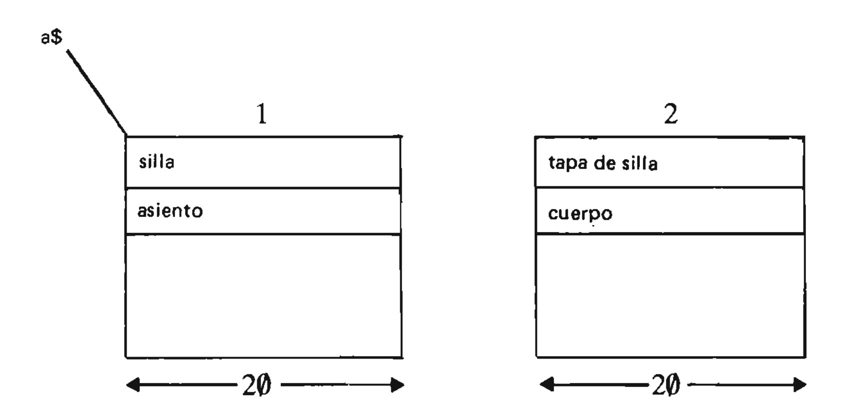
```
2680  LET i = j:LET col = 1:GOSUB rastreo
2690  LET ep1 = a
```

La subrutina *rastreo* devuelve a, que es el apuntador al último elemen-

to entrado que yo he transferido a ep1 (abreviatura de apuntador de fin 1).

```
2700  LET i = k:LET col = 2:GOSUB rastreo
2710  LET ep2 = a
```

Todo esto supone entonces que a$ va a aparecer así:



y se ha dimensionado con:

```
DIM a$(50,2,20)
```

(Puesto que el árbol es binario, el camino más largo de rastreo no puede ser mayor que la mitad de su número máximo de nodos, que nosotros establecemos en 100.)

Ahora, para buscar una no-coincidencia:

```
2720  IF ep1 = 0 OR ep2 = 0 THEN LET en línea = 1:GOTO 2750
2730  IF a$ (ep1,1) = a$ (ep2,2) THEN LET ep1 = ep1 – 1:
      LET ep2 = ep2 – 1:GOTO 2720
2740  LET en línea = 0
2750  LET distancia = ep1 + ep2
2760  PRINT "distancia ="; distancia
2770  IF en línea THEN PRINT "en línea"
2780  RETURN
```

De todo esto se encarga la rutina *rastreo*...

rastreo (3000)

Supongamos que tenemos una subrutina llamada *menosuno* en la que, dado un apuntador a un nodo, i, devuelva en m$ su madre y un apuntador a su madre en i nuevamente. También devuelve un 0 ó un 1 en raíz indicando si el nodo es o no es el raíz.

```
3000  LET a = 1:LET a$(a,col) = n$(i)
3010  GOSUB menosuno
3020  IF raíz THEN RETURN
3030  LET a = a + 1
3040  LET a$(a,col) = m$
3050  GOTO 3010
```

menosuno (3200)

Ahora, ya tenemos escrito *menosuno* que cubre todos los casos y propósitos. Es la segunda mitad de madre. Realmente madre debería llamar a *menosuno* (y así lo hace en la versión final de este programa), pero aquí presento esta codificación tal y como la concebí originalmente. Pienso que es importante conocer que, al igual que en cualquier problema de diseño de ingeniería, a veces se hace necesario quitar las ruedas, añadir un nuevo eje, y montarlas de nuevo en la forma inversa. Es fácil tener la impresión de que el profesional puede escribir con gran facilidad una codificación sin defecto de forma, lógica, correcta y elegante. Muéstreme alguien que diga que él puede hacerlo y le diré que está fanfarroneando.

Ahora me adentraré en el trabajo que nos ocupa:

```
3200  FOR r = 1 TO 100
3210  IF p(r,1) = i OR p(r,2) = i THEN LET m$ = n$(r):
      LET raíz = 0:LET i = r:RETURN
3220  NEXT r
3230  LET raíz = 1
3240  RETURN
```

¡Y ya está! Hemos obtenido un modo de escribir y medir la relación entre cualquier conjunto de elementos que pueden conformar una estructura tipo árbol.

SISTEMAS EXPERTOS

Lo que hemos estado discutiendo es una forma primitiva de un tipo de programa conocido como un sistema experto.

La idea es que un humano experto en alguna materia facilita al ordenador el conocimiento sobre su disciplina con el tipo de solución que hemos estado discutiendo. Los gráficos creados en la práctica son más complejos que los árboles que hemos estado utilizando. Con toda probabilidad, existirán enlaces secundarios entre los nodos. Por ejemplo, en el árbol de muebles, "patas de banqueta" está a una distancia de 4 de "patas de silla" aunque ambos tienen la característica "patas"; por tanto existe un caso para crear un enlace directo entre ellos (y, por supuesto, otro a "patas de mesa"). El tratamiento de estos enlaces extra presenta otros problemas de programación adicionales, por lo cual he preferido ignorarlos hasta ahora.

Para que este sistema sea de utilidad, no sólo debe ser capaz de memorizar conocimiento, sino de hacerlo de forma apropiada para el usuario. Por ejemplo, yo podría construir una base de conocimientos médicos sobre enfermedades contagiosas basado en los mismos principios que los que he destacado. Puedo incluso utilizar exactamente el mismo programa, excepto que el mensaje de diálogo de entrada "parte de" en la subrutina *crecer* se debería cambiar por "síntomas de", más apropiado.

Ahora puedo entrar cosas como:

> Los síntomas del sarampión son fiebre y granos que aparecen después de la fiebre.

Sin embargo, cuando tratamos, en este caso de recopilar el conocimiento, deseamos que el programa actúe como emisor de diagnósticos. En otras palabras, dados los síntomas de "fiebre" y "granos que aparecen después de la fiebre" queremos que la máquina responda triunfalmente "sarampión". En este ejemplo en particular esa conexión es sencilla de establecer. Expresado en términos de parámetros para nuestro programa sería:

> Si *conex* devuelve distancia = 2 AND NOT enlínea THEN devolver *madre* de un nodo de entrada.

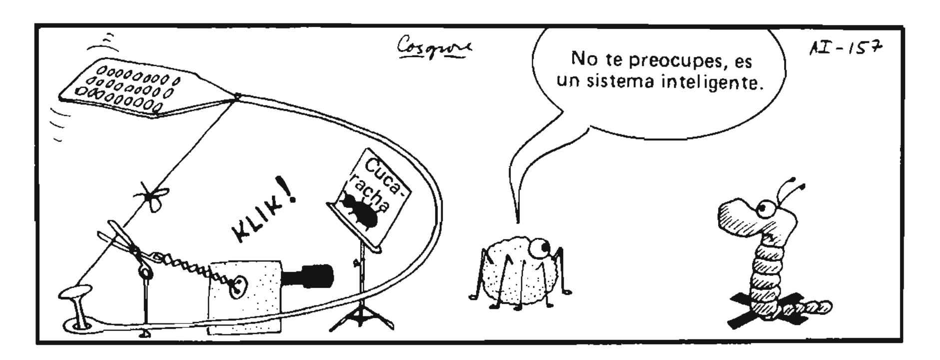
Podemos fácilmente imaginar situaciones más complejas, en las que no se ofrecen al programa datos suficientes para que trabaje. Supongamos que le digo simplemente que el paciente tiene fiebre. Podría recorrer el árbol buscando un nodo "fiebre", encontrar el otro nodo hija

correspondiente a él y luego preguntar si ese síntoma (granos después de fiebre) también está presente. En una base grande de conocimientos es posible que al usuario no le resulte obvio el *porqué* el programa efectúa esa pregunta; por tanto es importante también que el usuario tenga la oportunidad de cuestionar el "razonamiento" del programa. En otras palabras, el usuario debería ser capaz de preguntar "¿Por qué?" y el programa podría responder "Porque, si es así, el paciente tiene sarampión". Por supuesto que la hoja que representa el síntoma sobre el cual está preguntando el programa puede estar a varios niveles por debajo en el árbol: la conexión en este caso no se presentaría tan obvia de cara al usuario.

En la vida real, todo esto no se encuentra tan claramente definido como nos gustaría. Lo más probable es que un conjunto de síntomas apunte a una enfermedad en particular con cierto grado de probabilidad en lugar de con una absoluta certeza. Por tanto, un sistema experto práctico incluiría algunos datos de probabilidad como parte del valor de cada nodo.

En los útlimos cinco años aproximadamente se han desarrollado una serie de sistemas expertos de gran potencia. Uno de ellos en particular, llamado PROSPECTOR, se ha utilizado para localizar yacimientos minerales, una vez que se le ha dado detalles de la estructura geológica. Otro sistema, llamado MYCIN, sugiere el tratamiento apropiado para los pacientes que sufren de infecciones bacterianas.

Por supuesto, al margen del esfuerzo de programación requerido para producir un sistema experto, el experto humano debe dedicar una gran cantidad de tiempo al programa creando la base de conocimientos. Cualquier tentativa sería implicará miles de horas y, dado que los expertos humanos no tienen una dedicación continuada y completa, llevará años completarlo. Todavía existe otro problema: ¿se ha percatado de la



lentitud de *búsqueda* al recorrer nuestro árbol? Recuerde que la introducción de los datos sobre muebles le llevó a Vd. nada más que unos pocos minutos. Una estructura de datos práctica va a consumir casi todo el tiempo. Está claro que Vd. no escribiría un sistema *real* experto en BASIC y lo debería ejecutar en una máquina más bien potente con bastante memoria y un almacenamiento rápido de respaldo, pero incluso así... Al contemplar problemas de este tipo se explica porqué los usuarios de ordenadores estén siempre tratando de conseguir una mayor velocidad y una mayor potencia.

Proyectos

1. Las búsquedas a través de n$ son actualmente ineficaces ya que los 100 elementos se examinan siempre aunque el árbol no esté lleno. Revise el programa para eliminar este problema.
2. La mayoría de los manuales de mantenimiento de coches describen las averías y sus probables causas al final de cada capítulo. Utilizando estos datos (u otros semejantes) y *crecer*, cree Vd. un árbol de diagnóstico apropiado. Luego revise *búsqueda* para que realice los diagnósticos de una forma más adecuada.
3. En este momento, el usuario tiene que forzar su visión del árbol según un patrón binario, ya que *crecer* hace hincapié en que cada nodo madre tiene sólo dos hijas. Sin embargo, el hecho de que esto se pueda hacer siempre, significa que esto lo podría hacer *crecer* en lugar del propio usuario. Modifíquelo apropiadamente. La rutina *crecer* tendrá que utilizar un conjunto de nombres de nodos arbitrarios aunque distintos para los nodos falsos que cree. Esto se hace fácilmente si se dispone de un contador (digamos, d) inicializado a 1, dando al nodo falso el valor "dn" + STR$(d) e incrementando luego d, de forma que los nodos falsos sean dn1, dn2, etc.
4. El proyecto 3 sugiere una modificación útil para la rutina *conex*. Actualmente *conex* cuenta las ramas que incluyen los nodos falsos, los cuales, según he comentado anteriormente, pueden dar una visión parcial de la distancia entre dos nodos. Puesto que cada nodo falso comienza ahora con "dn", (garantizando con las pruebas necesarias que los demás nodos no comienzan con dn), sería relativamente fácil disponer que el proceso de cómputo ignore la rama de entrada o salida a/desde un nodo falso (*no* ambos, piénselo bien) para obtener una medida de la distancia más precisa.
5. Proporcione a Liz un "área de experiencia" injertando un árbol de conocimientos en él. Luego permita que utilice esto para efectuar cambios sutiles en el tema de conversación o sustituir un término genérico por uno específico.

9
Juego

Una de las áreas de la investigación sobre inteligencia artificial que más éxito ha tenido en los últimos treinta años, se ha desarrollado sobre el diseño de programas para juegos frente a un contrincante humano. Esto no debe sorprender ya que, después de todo, en el tipo de problemas que hemos examinado hasta ahora era necesario aportar a la máquina gran cantidad de información (un modelo de universo) para dar sentido a los datos que se le introducían. Un tablero de juego representa todo un mundo en pequeño, por lo que es relativamente sencillo contar al ordenador con exactitud todo lo que hay que saber. Aun así, en los juegos especialmente atractivos como el ajedrez, los mejores jugadores no saben todo lo que hay que saber, por lo que el modelo del ordenador es necesariamente incompleto. En realidad, ésta es una de las condiciones para que un juego sea atractivo. Si los jugadores lo conocen completamente, saben la solución al primer movimiento efectuado. Esta es la idea que se esconde detrás de la broma que supone que un programa de ajedrez sumamente potente, jugando con las negras, responda a un movimiento de apertura peón a reina 4 reflexionando durante cuarenta y cinco minutos para concluir luego abandonando.

ARBOLES DE JUEGO

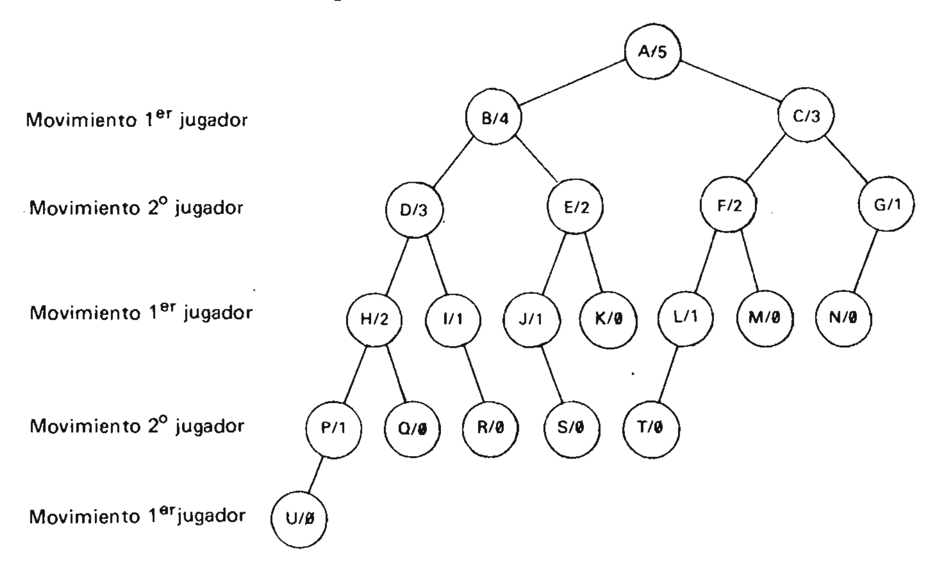
Se puede ilustrar el curso de un juego mediante un árbol. Examinemos un ejemplo sencillo. Definiré las reglas de un juego de esta forma:

1. Existen dos jugadores que se alternan para mover.
2. El estado inicial del juego es que en el tablero existen cinco bastoncillos.

3. Un movimiento autorizado consiste en eliminar uno o dos bastoncillos.
4. Un jugador vence cuando el otro jugador elimina el último bastoncillo.

Probablemente se habrá dado cuenta de que este juego es una forma simplificada del juego *Nim* (en el que hay más bastoncillos y más opciones). Si ha jugado alguna vez con Nim sabrá que es un juego muy simple.

El árbol muestra cada posible posición en el juego, mostrando los enlaces entre las posiciones posibles sucesivas. Las letras de los nodos son simples referencias, pero los números indican el número de bastoncillos dejados en esta etapa:



Observe que no he dicho nada todavía sobre si un movimiento es bueno o malo. Por ejemplo, si el juego llega al nodo E, el jugador 1 no va a quitar dos bastoncillos porque llegaría al nodo K y perdería. Pero éste sería un movimiento válido, es decir legal, y el árbol sólo trata todos los posibles movimientos legales.

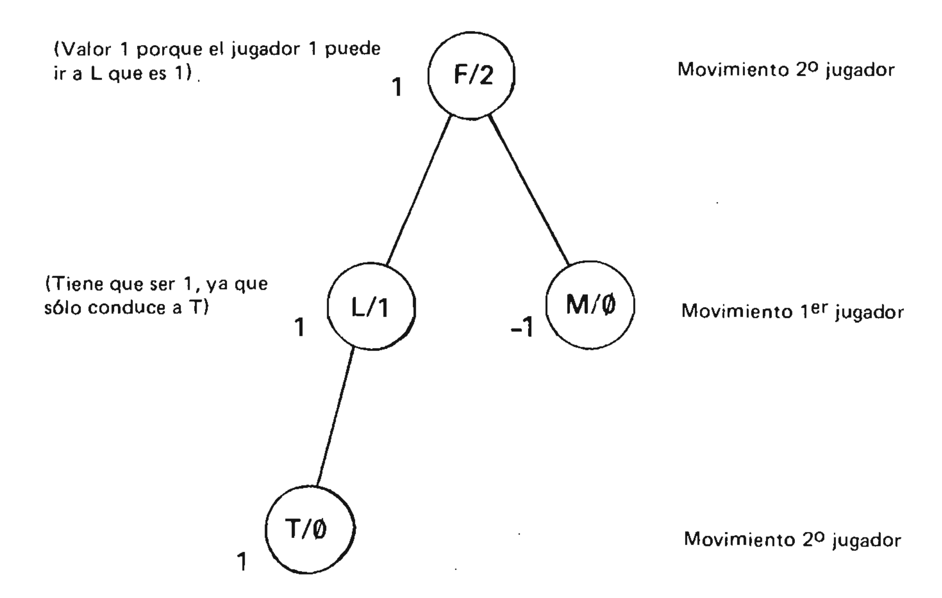
¿CUANDO UN MOVIMIENTO ES BUENO?

Asignemos ahora algunos valores a los nodos que indiquen la calidad de un movimiento en particular. Obviamente esto se hace más fácil en las hojas porque sabemos quién ha ganado. Definamos un valor 1 para indicar que gana el jugador 1 y un valor -1 para indicar que gana el ju-

gador 2. (Yo he utilizado −1 por sentido de simetría, pero Vd. *podría* escoger el 0.)

Por ejemplo, el nodo U tiene el valor −1 porque el jugador 1 toma el último bastoncillo. De manera análoga, Q, R, S y T están puestos a 1 y K, M y N a −1. En algunos casos, se pueden asignar valores a los nodos que no son hojas. Por ejemplo, como el nodo P sólo tiene una rama al nodo U debe tener el mismo valor que el nodo U (es decir −1). Esta es otra forma de decir que una vez el juego ha llegado al nodo P, el jugador 2 tiene garantizada la victoria.

Ahora asumiremos que cada jugador efectúa su mejor movimiento posible en cada jugada. Esto significa que si el jugador 1 tiene la opción de ir a un nodo cuyo valor es 1, lo tomará, por lo cual el nodo a partir del cual está moviéndose se puede decir que tiene un valor 1. Análogamente, el jugador 2 intentará llegar a un nodo −1. Utilizando esta regla, podemos "recorrer" el árbol hacia arriba partiendo de las hojas, evaluando cada nodo según se efectúa el recorrido. Examine una pequeña zona del árbol:



Si recorre Vd. el árbol desde abajo hacia la raíz, verá que la raíz se valora en 1. Dicho de otra forma, ¡el primer jugador puede ganar siempre!

Esto nos demuestra que Nim no es un juego atractivo. Además, los juegos para los que se puede escribir un conjunto completo de movi-

mientos no pueden considerarse como atractivos, ya que de ser así el juego se puede realizar de modo totalmente automático, sin posibilidad de adivinanza o inspiración. Por otra parte, esta peculiaridad es la que ofrece al ordenador más facilidad para el juego.

De hecho, es un ejercicio relativamente sencillo dejar que el ordenador se cree su propio árbol con arreglo a las reglas del juego, evaluando cada nodo sobre la marcha y utilizando esta información para vencer a un jugador humano, en el supuesto, claro está, de que juegue él primero. Dejaré esto como un problema sobre el que deberá Vd. reflexionar. Nuestra experiencia en árboles quizás le sea útil.

En este capítulo, estoy más interesado en investigar las técnicas para el tratamiento de juegos más complejos para los que no se puede construir árboles completos, sea debido a que la memoria disponible es insuficiente, sea debido a que llevaría demasiado tiempo, o a la conjunción de ambos.

Es fácil observar que los árboles quedan fuera de control rápidamente si Vd. considera uno para el ajedrez. Las piezas blancas tienen veinte posibles movimientos de apertura; es decir, dos posibles movimientos con cada peón y dos con cada caballo. A partir de cada uno de esos veinte nodos, las piezas negras pueden realizar cualquiera de los mismos veinte movimientos. Es decir, 421 nodos después de un solo movimiento de cada uno. Se han efectuado estimaciones sobre el tiempo que llevaría jugar una partida de ajedrez ejecutando un árbol total. Yo me inclino por presumir que un Cray 1 (posiblemente la máquina más rápida) se encontraría hoy por el movimiento número diez, si hubiera estado trabajando en el problema desde los tiempos del Big Bang.

FUNCIONES DE EVALUACION

La solución al problema es agrandar el árbol solamente una pequeña porción desde cada posición según va surgiendo. Sin embargo, esto presenta una nueva dificultad. Los únicos nodos con valor *cierto* son las hojas y si solamente hacemos crecer al árbol en un pequeño trozo, probablemente no alcancemos ninguna. Lo que necesitamos es una medida de la calidad de una posición con la que podemos señalizar un nodo, que no siendo una hoja puede estar tan lejos como seamos capaces de alcanzar a ver en este movimiento.

A esta medida la llamaremos *función de evaluación* que deberá escogerse con mucho cuidado para que sea satisfactoria para un determina-

do juego. Por ejemplo, piense en una función de evaluación sencilla para el ajedrez. Es algo más complejo que el simple cómputo de piezas. Podríamos asignar un valor a cada pieza (peón = 1, caballo = 3, alfil = 4, torre = 6, reina = 10, por ejemplo) y luego evaluar simplemente el poder de las blancas en base a estas premisas. Así, si se tienen 3 peones, los dos caballos y una torre, tendrá $3 \times 1 + 2 \times 3 + 6 = 15$. Sin embargo, si esto es bueno o malo dependerá de las fuerzas de su oponente; podríamos entonces restar su valor ponderado calculado de igual forma. Si las negras tienen 2 peones, un caballo, una torre y su reina, su valor será de $2 \times 1 + 4 + 6 + 10 = 22$ y la función de evaluación nos da $15 - 22 = -7$. Por tanto, al igual que para Nim, el primer jugador se procura el valor más alto posible mientras que el segundo intenta crear uno bajo. La diferencia es que ahora tenemos ciertas sombras de duda respecto al proceso. Por ejemplo, las blancas preferirían -3 a -7 aunque ninguna de ambas evaluaciones se puede considerar buena desde su punto de vista. También sucede que con una función tan simple el sacrificio de una reina nunca es beneficioso, aunque los jugadores de ajedrez reconocen este hecho como una empresa válida en determinadas circunstancias. En cualquier caso no es probable que la función conduzca a una jugada de jaque-mate, toda vez que esta jugada no se considera. Así pues, se puede imaginar un programa basado en esta idea que acumule muchas piezas de ventaja, errando en su objetivo final al no aprovechar su superioridad. Los programas modernos de ajedrez tienen, obviamente, funciones de evaluación mucho más sofisticadas que tienen en cuenta cosas como la bondad en el control de las filas centrales, cómo se protege al rey, la calidad de la formación de los peones, etc. Incluso así, generalmente, son insuficientes al llegar a la parte final del juego.

EL EFECTO HORIZONTE

Todos los árboles de juegos desarrollados parcialmente padecen de un serio defecto. Recuerde el sacrificio de la reina que mencioné anteriormente. Si se evalúa ese movimiento, con toda seguridad se conceptuará como un mal movimiento. Sin embargo, el movimiento siguiente puede que sea la emboscada para un jaque-mate. Si se detiene el proceso de crecimiento del árbol antes de que se haya examinado la posición de jaque-mate, el programa nunca llegará a saber que se llegó a esa situación. Es como si se mantuviera "sobre el horizonte". En consecuencia, el programa descartará la opción de perder su reina.

Naturalmente, los jugadores humanos hacen algo parecido. Ellos pueden ver además una serie de movimientos posteriores con cierta con-

fianza, pero tienden a tener una visión menos rígida sobre el valor de las piezas y por lo tanto pueden sentirse deseosos de examinar opciones extremas. En realidad, son más propicios al riesgo. Es difícil construir en un programa este tipo de posturas. Es más difícil aún contemplar en un programa otro atributo muy común en los seres humanos durante el juego: las fanfarronadas. Incluso en el ajedrez, los humanos utilizan este tipo de trucos sutiles. Un jugador efectúa un movimiento aparentemente intrascendente, confiando en que su oponente reaccionará sin detenerse a pensar demasiado en la jugada. Pero en el póker, por ejemplo, las fanfarronadas, las baladronadas, constituyen quizás el 80% del juego. Aquí sí puede Vd. confiar en que el ordenador sea especialmente bueno. Después de esto puede evaluar todas las probabilidades necesarias con más efectividad que el elemento humano y con una cara muy especial de póker. Sin embargo, no existen *señales* como tics nerviosos, miradas confiadas o caras sudorosas que puedan detectar sus oponentes humanos. E incluso aunque pudiéramos construirlas dentro del programa, sería difícil convencer a los jugadores humanos de que al ordenador le preocupa perder una jugada.

OTHELLO

Si Vd. desarrolla un jugador NIM en BASIC, verá que no es ágil en absoluto. Ya hemos visto en nuestras investigaciones anteriores sobre el crecimiento y búsqueda sobre un árbol que se pierde mucho tiempo esperando a que BASIC piense en el problema. Así pues, la implantación de cualquier juego serio es algo anodino a menos que lo vaya a realizar en código de máquina o en un lenguaje complicado de alto nivel, como Pascal. Incluso así, con un procesador relativamente lento como un Z80, si el programa contempla cinco o seis bazas o movimientos por adelantado, puede tardar varios minutos en efectuar un movimiento y para entonces su oponente humano se ha olvidado de lo que iba a hacer o se ha aburrido y se ha marchado.

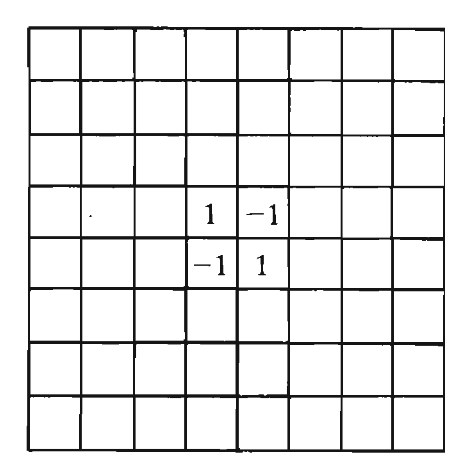
Por estas razones, no voy a implantar un juego completo para dos personas, en BASIC. En su lugar, discutiré algunos de los problemas (y sus posibles soluciones) de un juego en particular, Othello, y escribiré algunas de las rutinas más importantes en BASIC a modo de ilustración. ¿Por qué Othello? Bien, en primer lugar, las reglas de este juego son sumamente sencillas y el ordenador dispone de ciertas ventajas, ya que un jugador humano no puede ver fácilmente las jugadas futuras con anticipación. También sucede, como veremos más tarde, que las funciones de evaluación son bastante fáciles de escribir.

EL TABLERO

Lo primero que necesitamos es una estructura interna para el tablero de 8 X 8 de Othello. Realmente vamos a necesitar un gran número de ellas, ya que existirá una por cada nodo del árbol. Así tendríamos:

```
DIM b (50, 8, 8)
```

que permite un máximo de 50 nodos. Esto ocuparía 50 X 8 X 8 X 5 = = 16K. Existen varios métodos para acortar esta cifra, según veremos más adelante. Nos referiremos a los jugadores como "1" y "–1"; así pues, la configuración inicial del tablero es:



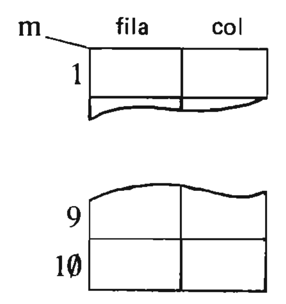
			1	–1			
			–1	1			

La ventaja de esto es que se pueden efectuar referencias a "jugador" y "–jugador" de forma que una sola rutina pueda tratar los movimientos, sea cual fuere el jugador.

EL GENERADOR DE MOVIMIENTOS LEGALES

El problema más engañoso de Othello es listar todos los movimientos legales para un jugador. He dividido el problema en dos partes. La primera, anotar las posiciones donde el jugador actual se encuentra adyacente a una pieza del oponente (–jugador). Por supuesto que esto sólo es legal si existe una pieza "jugador" al final de una línea recta trazada entre estos dos puntos. Así pues, le llamaré un "generador de movi-

mientos para-legales" y anotaré su resultado en un array llamado m, organizado de esta forma:



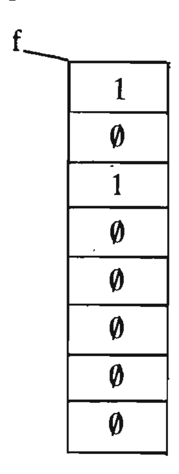
Esto permite al programa encontrar un máximo de diez movimientos para-legales para posterior examen.

El generador de movimientos legales examina cada uno de estos movimientos y decide si son *completamente* legales o no. Para ello tiene que examinar el tablero en cada una de las ocho direcciones, que en realidad son los puntos cardinales Norte, Noreste, Este, Sureste, Sur, Suroeste, Oeste y Noroeste. Sería aburrido y tedioso tener que escribir ocho rutinas distintas, especialmente porque sólo se diferencian en los valores incrementales de fila y columna necesarios. Por tanto, utilizaremos un array i que especifique dichos incrementos:

i

fila	col	
−1	Ø	Norte
−1	1	Noreste
Ø	1	Este
1	1	Sureste
1	Ø	Sur
1	−1	Suroeste
Ø	−1	Oeste
−1	−1	Noroeste

Una rutina llamada *prumov* se encarga de examinar cada una de estas posibilidades y devuelve un "1" en el elemento correspondiente de un array llamado f si esa dirección en particular conduce a la captura de piezas. Así, por ejemplo, si un movimiento captura piezas en dirección Norte y Este, f aparecerá como:



Ahora, si la suma de todos los valores en f es cero, el movimiento no es legal. Por otro lado, podemos realizar el movimiento y f nos indica en qué direcciones podemos realizar capturas. Por tanto, la rutina *generar movimientos legales* será algo parecido a:

```
1000  DIM f(8)                      (inicializar f)
1010  LET movimiento = 0            (crear un contador de movimientos)
1020  GOSUB gplm                    (generar movimientos para-legales)
1030  FOR p = 1 TO 10
1040  IF m(p,1) = 0                 (porque no hay más movimientos
      THEN RETURN                   para-legales)
1050  FOR d = 1 TO 8          )
1060  GOSUB prumov            }     probar movimiento en las 8
                                    direcciones
1070  NEXT d                  )
1080  LET sf = 0                    (inicializar suma de f)
1090  FOR d = 1 TO 8          )
1100  LET sf = sf + f(d)      }     calcular suma de f
1110  NEXT d                  )
1120  If sf > 0 THEN GOSUB          (si el movimiento es legal, hágalo
      mover:LET movimiento =        y cuéntelo)
      movimiento + 1
```

```
1130  If mover > 4 THEN        (así el programa permite cinco
      RETURN                   movimientos a examinar como
                               máximo)
1140  NEXT p
1150  RETURN
```

Generar movimientos para-legales (gplm)

```
1200  DIM m(10,2)
1210  FOR fila = 1 TO 8
1220  FOR col = 1 TO 8
1230  LET a(fila + 1,col + 1) = b(n,fila,col)
1240  NEXT col
1250  NEXT fila
```

El segmento de rutina anterior inicializa m y luego toma el nodo actual de b (el enésimo) y transfiere el estado del tablero a "a". Sin embargo "a" está dimensionado a 10 X 10, y solamente se transfieren las 8 columnas y filas centrales. De esta forma, cuando se ejecutan las pruebas sobre el cuadrado de una fila más arriba o una columna más adelante, no hay necesidad de ver si existe tal fila o columna, ya que yo he puesto un borde alrededor del tablero para asegurar que esto sea así.

```
1260  LET p = 1
1270  FOR fila = 1 TO 8
1280  FOR col = 1 TO 8
1290  IF a(fila,col) = 0 THEN GOSUB adyacente
1300  IF ady THEN LET m(p,1) = fila;LET m(p,2) = col:
      LET p = p + 1
1310  IF p > 10 THEN RETURN
1320  NEXT col
1330  NEXT fila
1340  RETURN
```

Esta rutina examina simplemente cada punto del tablero buscando un espacio. Cuando lo encuentra llama a una subrutina llamada *adyacente*, que comprueba si un cuadrado adyacente se encuentra ocupado por una pieza del componente. Si es así, devuelve una señal ady como 1; en caso contrario ady es cero. Si ady es 1, esta posición se suma a m, hasta que haya diez movimientos para-legales que escoger o hasta que no quede ningún movimiento para-legal; lo primero que suceda.

prumov

Esta subrutina *prumov* tiene que decidir primero dónde mirar y qué incrementos utilizar:

```
1400  LET fila = m(p,1):LET col = m(p,2)
1410  LET incr = i(d,1):LET incc = i(d,2)
```

Luego va a lo largo de esta línea, buscando un valor "jugador" para rodear los valores "–jugador" con:

```
1420  LET fila = fila + incr:LET col = col + incc
1430  IF col > 8 OR fila > 8 THEN LET f(d) = 0:RETURN
1440  IF b(n,fila,col) = jugador THEN LET f(d) = 1:RETURN
1450  GOTO 1420
```

mover

Esta rutina utiliza la misma información para cambiar todos los valores "–jugador" a "jugador". Sin embargo, lo hace para las ocho direcciones:

```
1600  LET fila = m(p,1):LET col = m(p,2)
1610  FOR d = 1 TO 8
1620  LET incr = i(d,1):LET incc = i(d,2)
1630  IF NOT f(d) THEN GOTO 1660
1640  LET fila = fila + incr:LET col = col + incc
1650  IF b(n,fila,col) = –jugador THEN LET b(n,fila,col)
      = jugador:GOTO 1640
1660  NEXT d
1670  RETURN
```

adyacente

Lo cual nos lleva a la rutina más sencilla, la rutina *adyacente*:

```
1800  LET ady = 0
1810  FOR x = 1 TO 8
1820  LET incr = i(x,1):LET incc = i(x,2)
```

```
1830 IF a(fila + incr,col + incc) = -jugador THEN LET ady = 1
1840 NEXT x
1850 RETURN
```

y esto es todo. Así pues, el generador de movimientos legales necesita simplemente que se le pase n, que es el número del nodo a examinar, además del jugador que será 1 ó −1 dependiendo de quien sea su turno de juego.

La función de evaluación

Esta es una de las características mejores de un programa Othello. La función de evaluación más simple es la suma de los valores de los cuadros del tablero.

Así:

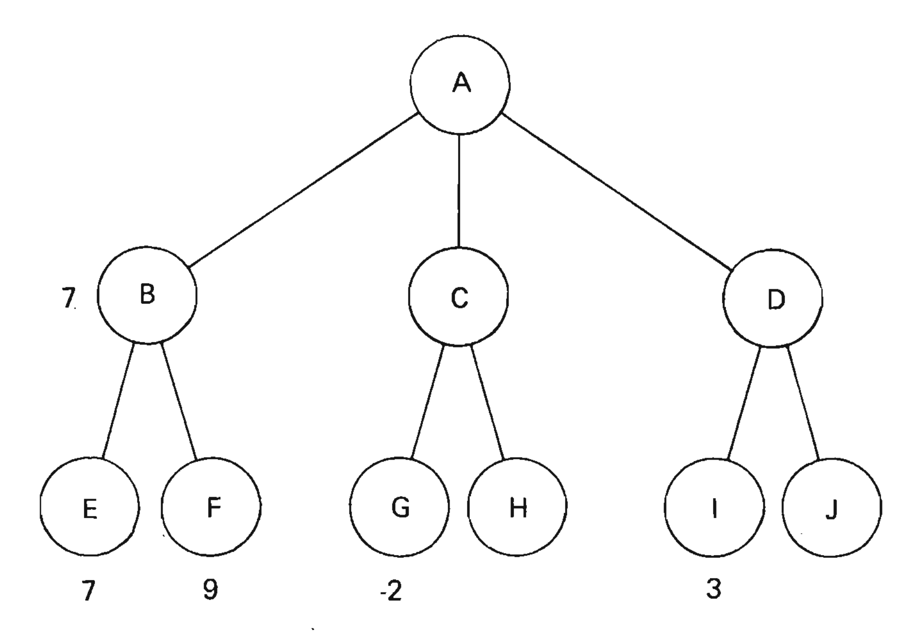
```
LET ef = 0
FOR fila = 1TO 8
FOR col = 1 TO 8
LET ef = ef + b(n,fila,col)
NEXT col
NEXT fila
```

dará, por ejemplo, ef = 4 si hay seis 1 (unos) y dos −1, mostrando así la ventaja mantenida por el jugador 1. Sin embargo, vea la sección Proyectos, donde encontrará algunas ideas adicionales.

LA PODA DEL ARBOL

Incluso un árbol como el que he descrito, crecerá a una velocidad alarmante. En algún momento en medio del juego, tenemos un nodo raíz, 5 nodos que parten de él, 5 nodos de cada uno de ellos (31 hasta ahora) y si asignamos suficiente espacio al array, otros 125 en el siguiente nivel. Y todos ellos a pesar de que he limitado el crecimiento del árbol de forma artificial considerando sólo 5 ramas en cada paso.

Sin embargo, es posible una solución más sensata que consistirá en la poda del árbol. Supongamos que llegamos a una parte del árbol con las siguientes características:



Supongamos que en el nodo A le toca su turno al jugador 1, por tanto va a tratar de aumentar al máximo la puntuación. Si los nodos E y F se evalúan como se muestra y se respaldan, B tiene el valor 7 (porque le toca su turno desde B a E o F al jugador −1 y tratará de reducir al mínimo la puntuación). Supongamos ahora que evaluamos el nodo G en −2. Con seguridad, el jugador −1 preferiría éste al 7, que es lo mejor que puede hacer desde B; por tanto no hay necesidad de evaluar H.

Desde el punto de vista del jugador 1, C es una elección desacertada porque el jugador −1 lo puede hacer por lo menos tan bien como para obtener un −2. De igual forma, D no es bueno porque el jugador −1 puede mejorar el 7 de B, bifurcando a I. Por tanto, no necesitamos J para nada.

A este I que actúa como un límite, le llamaremos alfa. Lo que hemos dicho hasta este momento es que tan pronto como llegamos en la jugada la reducción de puntuación a un valor menor que éste, no necesitamos examinar ya ningún otro nodo hija (o sus descendientes).

Ahora invertiremos todo el argumento. En una jugada de aumento de la puntuación, al llegar a un valor mayor que un límite beta, podemos frenar la marcha.

Este proceso se le conoce como poda alfa-beta, término éste bastante grandilocuente. Es un modo común y potente de reducir el espacio de búsqueda.

Proyectos

1. A pesar de que ejecutarlo tardará una eternidad, intente combinar el generador de movimientos, la función de evaluación y las técnicas de desarrollo y búsqueda de árboles del capítulo anterior para producir un jugador Othello que funcione. Si desea una competición real, pruebe con una versión en código de máquina, ¡no es tan difícil!
2. La función de evaluación propuesta es más bien primitiva. Una función mejor tendría en cuenta que es una mala táctica tomar un cuadrado adyacente a una esquina, ya que esto puede permitir que su oponente entre en la esquina donde no puede ser atacado. Igualmente, cualquier cuadrado de las filas y columnas 2 y 7 no son aconsejables. Revise la función para que tome esto en consideración.
3. Podríamos incrementar el espacio de búsqueda disponible si no tenemos cada posición del tablero registrada en cada nodo. Todo lo que necesitamos realmente es el movimiento que conduce a esa posición del tablero. Por otro lado, se duplica así alguna evaluación y el proceso, en consecuencia, se retarda de nuevo. De cualquier modo, inténtelo.

10
Proceso de imagen

Los capítulos dedicados a reconocimiento de patrones nos han mostrado que dado un conjunto de patrones que son suficientemente semejantes, es relativamente fácil diseñar programas que los distingan de otros patrones.

Ya he señalado que el término "patrón" no se aplica solamente a las imágenes o formas visuales; sin embargo, en este capítulo *voy* a ceñir nuestra discusión a las imágenes visuales.

El problema que tenemos con el reconocimiento de patrones visuales es que los seres humanos tienden a reconocer imágenes, formas que son realmente diferentes, como si pertenecieran a la misma clase. Por ejemplo:

5 5 5 5 5

Todos ellos son cincos, ¿o no lo son? Y sin embargo existen diferencias significativas entre ellos. Tres de ellos están formados por líneas continuas, otro tiene un vano entre dos líneas y el quinto tiene dos vanos. Cuatro de ellos tiene trazos horizontales, mientras que el otro tiene un trazo diagonal. Todos menos uno tienen una línea vertical. Sus tamaños son diferentes y uno de ellos tiene los trazos más gruesos que los otros cuatro.

Es interesante observar que quizás Vd. no se había percatado de todas estas diferencias antes de haberlas expuesto. ¿No necesita Vd. volver a mirar la figura para confirmar que lo que yo he dicho es cierto? Seguramente sí.

Dicho de otra forma, parece que se necesita un proceso inicial sobre la imagen formada en nuestra retina antes de intentar reconocerla. Esto es lo que quiero expresar bajo el término "proceso de imagen". Sus aplicaciones no están restringidas a los sistemas de reconocimiento de patrones a pesar de que ése es el contexto dentro del que se centra nuestra discusión. Por ejemplo, las soberbias imágenes obtenidas por la NASA de Saturno y sus anillos no aparecen verdaderamente claras hasta que pasan por un proceso de mejora de calidad con ayuda de un ordenador o, como yo diría, de "proceso de la imagen".

Así pues, este capítulo trata de las técnicas que le permiten crear formas estándares a partir de un amplio rango de entradas. Sin embargo, tenga en mente que no todas son aplicables en cualquier circunstancia. Por ejemplo, examinaremos una rutina que eliminará las discontinuidades presentes en dos de nuestros "cincos". Obviamente, esto es algo útil, a menos que el patrón bajo examen fuera la fotografía de rayos X de una soldadura sobre una plataforma petrolífera. ¡Entonces existirán discontinuidades que serían de lo más interesantes!

ENGROSAMIENTO Y ADELGAZAMIENTO

Abordemos primero el problema de la discontinuidad. En realidad podríamos eliminar las pequeñas discontinuidades engrosando más la imagen con la que comenzamos. Tracemos una imagen de muestra para trabajar en gráficos de alta resolución.

```
10  PLOT 60,100
20  DRAW 0,31
25  PLOT 61,100
30  PLOT 60,135
40  DRAW 35,0
45  PLOT 61,134
46  DRAW 34,0
50  PLOT 60,100
60  DRAW 10,–30,–4,4
```

esto dibuja un "5" con un pequeño corte en la parte superior. Vd. podría escribir una rutina para hacerlo con la ayuda de un mando de juegos, pero es difícil conseguir buenas curvas.

Ahora dibujaremos una casilla rodeándolo y otra casilla vacía para albergar el resultado engrosado para comparar.

```
 70  LET xs = 50:LET ys = 60
 80  GOSUB casilla
 90  LET xs = 140
100  GOSUB casilla
110  STOP
```

casilla es sencilla:

```
1000  PLOT xs,ys
1010  DRAW 0,80
1020  DRAW 60,0
1030  DRAW 0,–80
1040  DRAW –60,0
1050  RETURN
```

Así obtendremos un rectángulo de 80 × 60 cuyo ángulo inferior izquierdo está en las coordenadas xs,ys.

MIRANDO A TRAVES DE LA VENTANA

Ahora escogemos una ventana de 3 × 3 pixels a través de la cual podemos examinar la imagen. En alguna posición de la casilla podemos ver algo parecido a:

```
0  1  1
0  1  1
0  1  1
```

cuando la ventana cubre parte del trazo vertical. Sustituimos el pixel central por la suma de los pixels de la ventana, en este caso 6. Repetimos así el proceso para cada pixel. A la rutina que realiza esto la llamaremos *suma*:

```
1400  FOR y = 62 TO 138
1410  FOR x = 52 TO 108
1420  GOSUB sumaven
1430  LET p$(y – 60,x – 50) = STR$ total
1440  NEXT x
1450  NEXT y
1460  RETURN
```

Aquí se suponen varias cosas. Lo primero que se asume es un borde limpio alrededor del carácter de la casilla, ya que x e y están limitadas a dos pixels de anchura desde el borde. Eso evita tener el borde mezclado con el carácter y efectuar preguntas innecesarias sobre dónde se encuentra el pixel central actual. En segundo lugar, se llama a una subrutina llamada *sumaven* que devuelve la suma, en total, para una ventana especificada. En tercer lugar, el resultado pasa a un array tipo cadena para ahorrar memoria, ya que cada entrada debe ser un dígito comprendido entre el 0 y el 9.

Así ahora tenemos:

```
1   LET sumaven = 1200:LET suma = 1400:LET casilla = 1000
2   DIM p$(80,60)
```

y *sumaven* es:

```
1200  LET total = 0
1210  FOR r = y – 1 TO y + 1
1220  FOR c = x – 1 TO x + 1
1230  LET total = total + POINT(c,r)
1240  NEXT c
1250  NEXT r
1260  RETURN
```

Ejecútela introduciendo "GOSUB suma" como mandato directo. Luego váyase y disfrute de varias partidas de golf o cualquier otro juego. ¡Tardará en ejecutarse una eternidad!

Cuando se haya ejecutado, imprima el contenido de p$ con:

```
For n = 80 TO 1 STEP – 1:PRINT p$(n):NEXT n
```

Observe que para conseguir la imagen, p$ debe imprimirse en orden inverso, ya que está formada desde abajo a arriba. Las diez filas superiores se encuentran reflejadas en la figura 10.1. La imagen de la parte alta del "5" está todavía clara, pero si incluimos todos los valores que no sean cero, queda engrosada y la discontinuidad ha desaparecido. Por otro lado, si incluimos solamente los pixels cuyos valores sean igual o mayores que 6, por ejemplo, la curva total se adelgaza hasta no existir y la discontinuidad se hace más patente.

Esto lo podemos ver escribiendo una rutina llamada *media* que dibuja los puntos dentro de la casilla de la parte derecha, con arreglo a un valor de entrada suministrado:

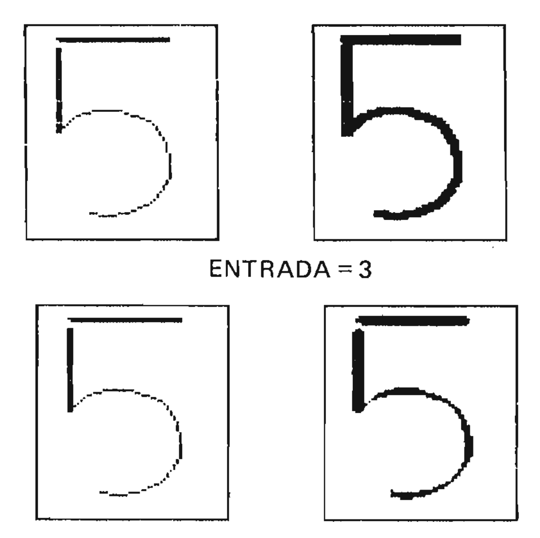
```
00000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000
000000012333333333333333333333333333333333333321000000000000
000000013566666666666666666666666666666666666642000000000000
000000013566666666666666666666666666666666666642000000000000
000000001233333333333333333333333333333333333321000000000000
00000001221000000000000000000000000000000000000000000000000
00000002442000000000000000000000000000000000000000000000000
00000003663000000000000000000000000000000000000000000000000
00000003663000000000000000000000000000000000000000000000000
```

Fig. 10.1.—Impresión de las diez filas superiores de p$. Tenga en cuenta que en la pantalla la visualización se desplazará de forma que la imagen no quedará clara.

```
1600 INPUT "entrada"; t
1610 FOR y = 62 TO 138
1620 FOR x = 142 TO 198
1630 IF VAL p$(y - 60,x - 140) > = t THEN PLOT x,y
1640 NEXT x
1650 NEXT y
1660 RETURN
```

Si prueba esto (utilizando GOSUB media, como mandato directo), con entrada de 1, 3 y 6 respectivamente, obtendrá los resultados mostrados en la figura 10.2. Así pues, seleccionando una entrada apropiada,

ENTRADA = 1



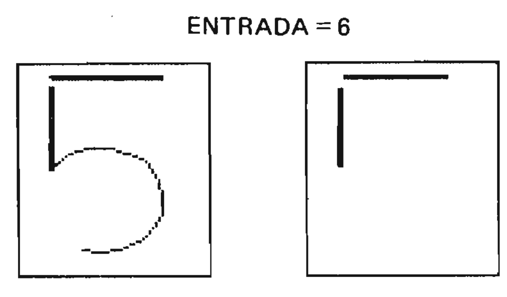
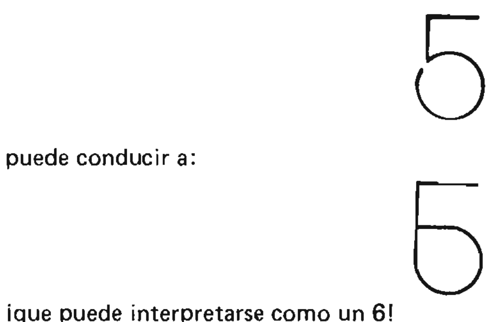


Fig. 10.2 — Resultado de utilizar diferentes entradas.

podemos hacer desaparecer las conexiones débiles o, por el contrario, crear conexiones donde anteriormente no existían.

Por supuesto que deberemos cuidar de no utilizar esta técnica de forma indiscriminada. Por ejemplo, el adelgazamiento de un "cinco" con esta imagen:



puede conducir a:

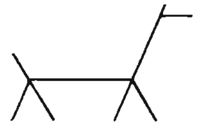
¡que puede interpretarse como un 6!

EL ESQUELETO DE LA IMAGEN

Así pues, con un valor de entrada bajo podemos eliminar discontinuidades no deseables. Desafortunadamente, al mismo tiempo engrosamos la imagen. En cualquier caso, probablemente comenzó siendo grueso (es decir, varios pixels de anchura), ya que lo que queríamos es que tuviera el grosor de una línea trazada a lápiz, para lo cual debe tener más anchura que un único pixel.

Idealmente, queremos crear un carácter estándar y la imagen esencial en la que estamos interesados es el *esqueleto* del carácter real que se nos presenta. Dicho en otras palabras, es la línea (o líneas) de un pixel de anchura la que nos describe la imagen.

Las imágenes complejas pueden identificarse de forma satisfactoria por su esqueleto (literalmente y metafóricamente). Es fácil ver que:

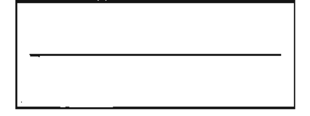


describe un perro o una criatura semejante (¿por qué no un gato?) sin ningún detalle que desordene el proceso de reconocimiento.

Sin embargo, la extracción del esqueleto de una imagen viene a ser como una ilusión o ficción trucada. Comencemos por examinar algunos de los problemas más espinosos con los que nos vamos a encontrar.

Problema 1: ¿Cuándo un borde es un final (y viceversa)?

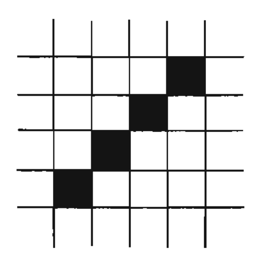
Piense en un simple rectángulo cuyo esqueleto he dibujado dentro de él:



Este caso es el más sencillo de tratar. Una posible solución sería cincelar los bordes hasta conseguir que quede una sola línea de grosor sencillo. Pero si esculpimos el perímetro total con igual criterio, la línea resultante será muy corta. Para obtener un esqueleto satisfactoriamente intuitivo necesitaremos tomar más de los lados que de la parte alta o la baja. Así pues, quizás deberíamos identificar los "puntos finales" y asegurarnos que éstos quedan intactos al cincelar los bordes. Pero ¿cómo encontramos los puntos finales?

Problema 2: ¿Cuándo una línea está conectada?

Supongamos que por un momento volvemos al Problema 1. Debemos detener la eliminación de material cuando la línea está próxima a perder su conexión. Si la línea contiene una sección diagonal como ésta:



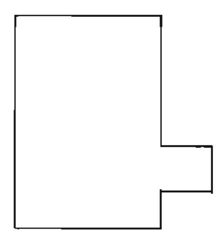
está ya conectada; tenemos el esqueleto. Pero si están conectados dos pixels diagonalmente adyacentes, ¿por qué los dos espacios a uno y otro lado de la línea no están conectados? ¡Si también tienen pixels diagonalmente adyacentes!

Problema 3: Simetría

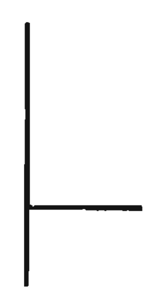
Si desarrollamos un algoritmo que trata un lado de la imagen de forma exactamente igual que el otro, cuando las líneas tienen dos pixels de grosor, se eliminarán ambos lados o no se eliminará ninguno.

Problema 4: Falsos puntos finales

Imagínese un bloque rectangular con un pequeño defecto:



Tenderíamos a considerar esto como un accidente, no como una parte significativa de la imagen. Sin embargo, podría ser tomado por nuestro algoritmo de punto final y retenido, dando:



puede ser bastante confuso.

UNA SOLUCION

Cuando hace unos pocos años me tropecé con este problema, estuve trabajando sobre ello cerca de un mes y a punto estuve de intentar algo más sencillo, como entrenar víboras venenosas, cuando cayó en mis manos accidentalmente un artículo de una revista científica que hacía referencia, de pasada, al hecho de que el cangrejo herradura tiene una singular característica en su sistema visual. Cuando una célula nerviosa de su retina reacciona ante una luz brillante, inhibe las restantes células próximas a aquélla para que no reaccionen. En otras palabras, si Vd. ilumina un círculo de luz en el ojo del cangrejo, él visualiza un punto de luz. ¡Es un algoritmo de obtención del esqueleto maravillosamente simple!

Esto me incomodó, ya que había pasado un mes investigando en un problema cuya respuesta conocía el cangrejo hace dos millones de años. Sin embargo, esto en sí mismo, no resuelve el problema completamente, ya que no conserva la conectividad. Para esto, necesitamos conocer algo sobre la densidad del patrón de la región del pixel que intentamos eliminar. La rutina *suma* que ya hemos tratado hará parte del trabajo, pero no hay razón para crear trozos de patrón que no existían allí anteriormente; por tanto sumaremos (*suma*) solamente aquellos puntos que no sean cero. Por ejemplo, el patrón:

```
        1
        1
        1
1   1   1   1   1   1
1   1   1   1   1   1
1   1   1   1   1   1
        1
        1
```

se transforma en:

```
        2
        3
        5
4   7   7   7   6   4
6   9   9   9   9   6 ← Ⓐ
4   7   7   7   6   4
        5       ↑
        2       Ⓑ
```

Ciertas cosas quedan claras inmediatamente. En primer lugar, un "2" ó un "3", no pueden eliminarse nunca, ya que en el ejemplo podemos ver que deben representar el final de una línea del esqueleto y un punto dentro de una línea del esqueleto respectivamente. Después de esto las cosas se van enturbiando. Por ejemplo, el "6" señalizado con A se debe preservar como final de una línea, mientras que el de B se debería eliminar y, como de hecho sucede, los 4 podrían quitarse, pero los 5 se deben mantener. Por supuesto, lo que hemos hecho ha sido crear una cierta escala de gris. (Ya mencioné en el capítulo 6 que llegaríamos a esto). Podemos imaginar que los 9 son más "luminosos" que los 2.

Ahora, ¿qué haría nuestro amigo el cangrejo? Bien, cuanto más luminosa sea la región que rodea a un pixel, más probabilidades tendrá de que se suprima. Si sumamos (*suma*) los valores luminosos de una ventana de 3 × 3 alrededor de B obtenemos 41. Alrededor de A el valor es de 35. Así, en alguna parte entre 35 y 41 hay un valor por debajo del cual deberíamos conservar un pixel cuyo valor sea 6. Existirán valores límites semejantes para los otros niveles de luminosidad. La tabla que doy a continuación nos los muestra:

Nivel de luminosidad	Valor por debajo del cual se preservará un pixel
9	78
8	66
7	52
6	40
5	30
4	22
3	siempre preservado
2	siempre preservado
1	siempre preservado

Ya hemos visto que se pueden eliminar los 2 y los 3 y que un "1" será un punto por sí mismo, por lo cual también se mantendrá (puede que sea el punto de una i).

ALGO DE CODIFICACION

Pongamoslo en práctica. Vamos a necesitar una rutina que ponga en la parte izquierda, la imagen engrosada de la casilla de la parte derecha, de forma que podamos utilizarla con las rutinas que ya disponemos. Escribiremos una rutina llamada *limpia* para inicializar ambas casillas y otra llamada *copia* para pasar de derecha a izquierda.

limpia (2000)

```
2000  FOR y = 62 TO 138
2010  PLOT INVERSE 1;52 + casnum * 90,y
2020  DRAW INVERSE 1;56,0
2030  NEXT y
2040  RETURN
```

Esto supone que se pasa un parámetro con el número de casilla a *limpia*, que se pone a cero para limpiar la casilla izquierda y a uno para limpiar la casilla derecha.

copia (2200)

```
2200  FOR y = 62 TO 138
2210  FOR x = 142 TO 198
2220  IF POINT (x,y) THEN PLOT x – 90,y
2230  NEXT x
2240  NEXT y
2250  RETURN
```

Por tanto *copia* asume que la casilla izquierda está limpia y copia sobre ella el contenido de la casilla de la parte derecha. Así pues, nuestro convenio es que todo el proceso tiene lugar de izquierda a derecha.

esquel (2400)

Lo primero será sumar solamente los pixels que están activados (on).

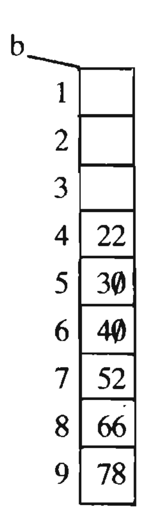
```
2400  DIM q$(80,60)
2410  FOR y = 62 TO 138
2420  FOR x = 52 TO 108
2430  IF POINT (x,y) THEN GOSUB sumaven:
      LET q$(y – 60, x – 50) = STR$ total
2440  NEXT x
2450  NEXT y
```

Observe que he dimensionado q$ de forma que se pueda limpiar rápidamente, pero eso significa que los pixels que quedan fuera del patrón estarán representados por "espacios" en lugar de ceros. Podría haber utilizado de nuevo p$, pero habría destruido el resultado de la suma que podemos necesitar más tarde y que lleva bastante tiempo conseguirla.

Ahora debemos examinar q$, decidiendo sobre la marcha los elementos a mantener:

```
2460  FOR y = 62 TO 138
2480  FOR x = 142 TO 198
2490  LET c$ = q$(y – 60, x – 140)
2500  IF c$ = " " THEN GOTO 2540
2510  IF VAL c$ < 4 THEN PLOT x,y:GOTO 2540
2520  GOSUB lumin
2530  IF br < b(VAL c$)THEN PLOT x,y
2540  NEXT x
2550  NEXT y
2560  RETURN
```

Esto necesita cierta explicación. Primero, los espacios se ignoran (línea 2500). Luego se dibuja cualquier pixel menor de 4. Cualquier otro pixel efectuará una llamada a una nueva subrutina, llamada *lumin*, que se encarga de evaluar el nivel de luminosidad local y lo devuelve en br. Este se utiliza para decidir si se dibuja el punto correspondiente y supone la existencia de un array b creado así:



Así, si por ejemplo br = 35 y c$ = "6", br es menor que b(6), por tanto se dibuja el punto. Sin embargo, si br es 41 y c$ = "6" el punto no se dibujará, tal como deseamos; b(1), b(2) y b(3) no se mencionan, por lo que no tienen porqué activarse.

lumin (2600)

Esta rutina es bastante parecida a sumaven, excepto que trabaja sobre q$:

```
2600  LET br = 0
2610  FOR r = y – 1 TO y + 1
2620  FOR c = x – 1 TO x + 1
2630  LET d$ = q$(r – 60, c – 140)
2640  IF d$ <> " " THEN LET br = br + VAL d$
2650  NEXT c
2660  NEXT r
2670  RETURN
```

Prueba de "esquel"

Ahora vamos a probarlo. Vd. puede crear un patrón con trazo grueso con el que trabajar, introduciendo:

```
GOTO 10: GOSUB media
```

No lo ejecute con RUN; perderá p$ y tendrá que ejecutar nuevamente *suma*. Luego:

```
LET casinum = 0:GOSUB limpia:GOSUB copia:LET casinum = 1:
GOSUB limpia
```

sitúe el "5" grueso en la casilla izquierda y limpie la casilla derecha. Finalmente:

```
GOSUB esquel
```

Obtendrá el resultado que aparece en la figura 10.3. Este es más delgado que el original, pero no es un esqueleto. Esto no le debe sorprender si piensa cómo funciona nuestro algoritmo. Elimina los pixels que no ofrecen problemas pero es muy conservador; ante la duda es precavido para que quede preservada la conectividad. La solución es obvia, repita el proceso:

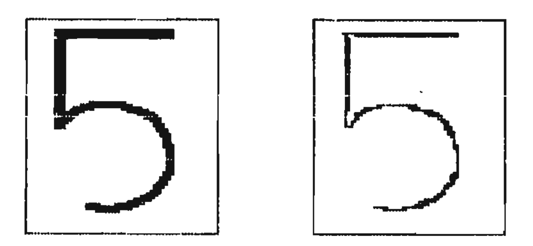


Fig. 10.3.—GOSUB esquel.

```
LET casinum = 0:GOSUB limpia:GOSUB copia:LET casinum = 1:
GOSUB limpia:GOSUB esquel
```

Ahora obtendrá la figura 10.4. Repita nuevamente el proceso y verá que no ha mejorado nada; así pues, no hay razón para seguir más adelante. Sin embargo, todavía no hemos conseguido un verdadero esqueleto. No obstante, observe cuidadosamente y verá que en conjunto, solamente nos han quedado líneas de grosor doble o sencillo. Por supuesto que las líneas de doble grosor aparecen por el problema de simetría que expusimos anteriormente y como ya dije entonces, no hay forma de que pueda eliminarlas ningún algoritmo. Ahora, sin embargo, estamos en distinta posición. Sabemos que solamente existen líneas de grosor doble

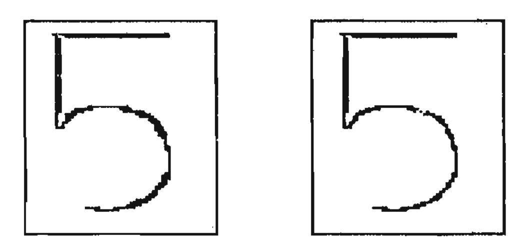


Fig. 10.4.–Suficiente GOSUB esquel.

o sencillo; por tanto se puede examinar la pantalla tratando de encontrarlas, borrando las duplicadas sobre la marcha.

líneadel (2800)

Escribiremos una rutina llamada *líneadel*. Para hacerlo:

```
2800  LET incx = 1:LET incy = 0:LET longitud = 0
2810  FOR y = 62 TO 138
2820  FOR x = 142 TO 198
2830  IF POINT (x,y) THEN GOSUB buscalon
2840  IF longitud = 2 THEN PLOT INVERSE 1; x – 1, y
2850  NEXT x
2860  NEXT y
```

Este segmento de codificación simplemente busca el comienzo de las líneas horizontales. Cuando encuentra una, llama a *buscalon* para que le diga la longitud que tiene. Si tiene dos elementos de longitud, se borra uno de los pixels.



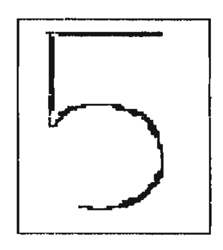
Tendremos que repetir esta operación ahora en dirección vertical y utilizaremos *buscalon* de nuevo, así que hay que pasarle la información sobre la dirección de la búsqueda de incx y incy:

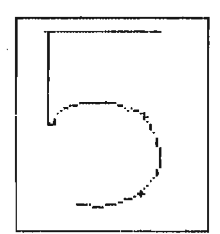
```
2870  LET incx = 0: LET incy = -1
2880  FOR x = 142 TO 198
2890  FOR y = 138 TO 62 STEP -1
2900  IF POINT (x,y) THEN GOSUB buscalon
2910  IF longitud = 2 THEN PLOT INVERSE 1; x,y + 1
2920  NEXT y
2930  NEXT x
2940  RETURN
```

buscalon (3000)

```
3000  LET longitud = 1
3010  LET x = x + incx
3020  LET y = y + incy
3030  IF NOT POINT (x,y) THEN RETURN
3040  LET longitud = longitud + 1
3050  GOTO 3010
```

Esto es más que suficiente. Incrementa simplemente la longitud hasta que encuentra un pixel vacío y vuelve. La figura 10.5 muestra su aplicación para nuestra figura previamente adelgazada. Verá que el resultado no es del todo perfecto. Existe una pequeña cruz cerca de la parte baja de la curva. Vea la sección de proyectos donde encontrará como mejorarlo.





Otras técnicas de proceso de imágenes

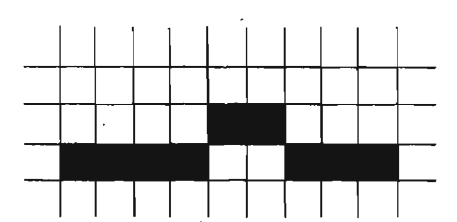
He presentado las rutinas de este capítulo como entidades más o menos individuales. Ha sido una política intencionada toda vez que

cada una de ellas consume mucho tiempo para ejecutarse en BASIC. Pretenden darle indicios sobre el tipo de técnicas empleadas.

Se podría escribir otro libro que tratara solamente de las técnicas para proceso de imágenes, pero debido en parte a la lenta respuesta del Spectrum y porque tenemos otras áreas interesantes que explorar, me conformaré con sugerirle algunas ideas más como proyectos.

Proyectos

1. La rutina *lineadel* tiene un error. Si una línea de dos pixels se conecta con otras dos líneas así:



   uno de los pixels se destruye y se pierde la conectividad. Escriba una rutina llamada *conectar* que devuelva un valor llamado nocon que sea un 1 si no existe conexión y cero en caso contrario. Entonces la línea 2840 (por ejemplo) se convierte en:

```
2840  IF longitud = 2 THEN GOSUB conectar
      IF nocon THEN PLOT INVERSE 1;x – 1, y
```

   Ahora podemos desembarazarnos de la cruz de 3 X 3 probando qué longitud será 2 ó 3.

2. Escriba una rutina que convierta la imagen en alta resolución que se ha procesado en una retina de 8 X 12 para que la reconozca el programa del capítulo 6.
3. Una alternativa para describir una imagen por su esqueleto consiste en eliminar los detalles mostrando solamente sus bordes. Esta técnica es útil para el análisis de escenas (tal como la visión de una sala por el ojo de un robot). Intente inventar un algoritmo para que haga esto.
4. Una faceta importante de la percepción visual humana (y animal) es el hecho de que *normalizamos* la imagen sobre nuestras retinas. Es decir, aunque se encuentre lejos un objeto, le vemos con la misma forma con independencia de que el tamaño de la imagen pueda variar terriblemente. Invente una rutina que, dada una imagen, la agrande para rellenar una "casilla" estándar sin distorsionar el resultado.

11
Proceso de la información y modelos biológicos

Comenzamos este libro con una exposición sobre la naturaleza de la inteligencia para ver si en determinadas circunstancias un sistema de ordenador se podía considerar capaz de ser artificialmente inteligente.

Tratamos esta idea de inteligencia de modo puramente *cualitativo* (en realidad, no estamos involucrados con el ordenador IQ o el MENSA, ¿o si lo estábamos?) y como recordará en todas nuestras discusiones, daba la impresión de que medíamos o evaluábamos la inteligencia comparándola con el comportamiento equivalente en los seres humanos. Todo esto nos sugiere que consideramos a los seres humanos como los *definidores* de la inteligencia o como los representantes del modo más avanzado de comportamiento inteligente. Esto a su vez implica suponer que el cerebro humano (que, al menos en este contexto podemos considerar como un cierto tipo de ordenador biológico) es el sistema de ordenador más apropiado y más capacitado para ejecutar procesos inteligentes.

Entonces nos podemos preguntar ¿por qué, a la vista de estas afirmaciones, no intentamos copiar la forma en que el cerebro humano lleva a cabo su proceso para construir así un sistema de ordenador inteligente? Esta es una pregunta engañosamente simple que tiene muchas respuestas interrelacionadas, una de las cuales se relaciona con la clara complejidad del cerebro y con el hecho difícilmente inseparable de que sólo se conocen parcialmente los mecanismos del cerebro. No es éste el lugar más adecuado para adentrarnos en una discusión pormenorizada sobre este tema pero, por todo ello, es una cuestión de gran trascendencia y en el curso de los años se han llevado a cabo intentos para tratar de imitar la forma en que el cerebro ejecuta sus asombrosas proezas de cálculo. Aunque hay que decir que dichos intentos se han enfocado generalmente sobre áreas de problemas específicos, se han obtenido im-

portantes logros y además hoy día existe una buena voluntad, por parte de los investigadores, de reconocer el valor de una aproximación interdisciplinaria a la ciencia del cerebro. Así, los psicólogos, los fisiólogos, los matemáticos, los informáticos e incluso los ingenieros electrónicos, se han dado cuenta, en algún momento, de que tenían que contribuir de alguna manera al avance sobre el conocimiento del ordenador más complejo y mejor dotado que existe.

En este capítulo haremos un breve examen sobre una posible solución para tratar de imitar las propiedades de cálculo del cerebro. Como verá, utilizamos algunos términos con cierto grado de libertad. Este examen lo realizaremos a un nivel muy bajo (en el sentido de concentrarnos sobre las propiedades de lo que quizás es la unidad más básica aunque gobernable de la estructura maravillosa del cerebro) a fin de ver cómo podemos describir sus propiedades de proceso de forma que se preste a una comparación con los sistemas de cálculo artificiales.

Antes de comenzar debemos decir que no pretendemos que esta aproximación nos lleve (o *pueda* llevarnos) directamente a la implantación de un procesador artificial "inteligente". Lo que intentaremos mostrar, en primer lugar, es cómo un sistema así tiende a poseer, en su propia naturaleza, alguna de las propiedades que pueden suponerse razonablemente que son la base de la inteligencia (las cuales, en consecuencia, se conectan con algunas de las ideas que se han presentado anteriormente), y en segundo lugar, intentaremos mostrar cómo podemos comenzar a extraer ciertas características como elementos clave para el diseño de los sistemas "artificialmente inteligentes".

PIEZA FUNDAMENTAL DEL PROCESO BIOLOGICO

Vamos a intentar percibir lo que acarrea nuestra aproximación y cómo podemos justificar nuestra metodología. Si estuviéramos intentando aprender algo sobre la estructura fundamental de un muro, por ejemplo, para poder constuir un edificio con propiedades análogas, podríamos identificar un *"ladrillo"* como la unidad más pequeña con la cual nos relacionamos. Probablemente, no nos interesa conocer cómo están hechos los ladrillos, ni en las propiedades químicas de la arena u otros componentes; más bien nuestro interés se centraría en las propiedades fundamentales del producto ya acabado. Aunque soy consciente de que esta analogía que me he ingeniado podría llevarnos a una discusión filosófica sin salida, todo lo que quiero sugerir es que una forma de comenzar a pensar sobre las propiedades de cálculo del cerebro es la de identificar sus "ladrillos" y comenzar a trabajar a partir de ellos. No nos haremos demasiadas preguntas sobre cómo trabaja este "ladrillo" bioló-

gico en términos de sus principales propiedades biológicas, pero nos concentraremos sobre la *función*, intentando descubrir solamente las características operativas que arrojen luz sobre nuestro tema inmediato de interés.

Sobre la base de esta analogía podemos identificar la célula biológica llamada *neurona* como la pieza de construcción fundamental del cerebro. Las neuronas se encuentran en muchas formas diferentes, por supuesto, pero podemos pensar en una célula típica (vea figura 11.1) cuyas propiedades se pueden describir simplemente como sigue...

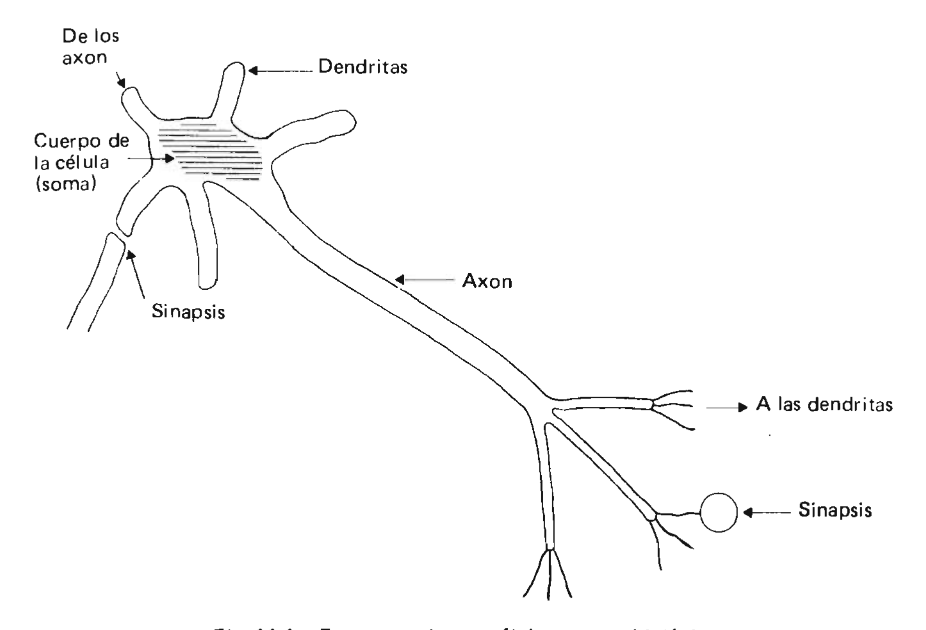


Fig. 11.1.—Estructura de una célula neurona biológica.

La célula tiene un conjunto de fibras llamadas *dendritas* que transportan información a la célula. Una fibra de salida de la célula transporta información sobre su estado actual y es capaz de pasarla a otras células con las que se encuentra conectada. Puesto que cada célula podría tener un gran número de dendritas (normalmente alrededor de 10.000) y dado que el cerebro contiene un número enorme de estas células (probablemente cerca de 10.000.000.000 ó así), se percatará de porqué el cerebro tiene una gran complejidad y tratar de conocerlo en todos sus detalles supone un desafío.

Ahora, desde nuestro punto de vista, es importante darse cuenta de que la información se transmite por las células neuronas como impulsos eléctricos. En otras palabras, una célula, en respuesta a algún patrón de actividad sobre sus entradas (dendritas), puede emitir un impulso de bajo voltaje que viaja a lo largo de su fibra (axon) de salida y pasa esta información a todas las células con las que está conectada. Cuando se emite un impulso en este forma decimos que la célula "dispara", indicando con este término que existe una actividad de la célula. Sin embargo, este disparo es un proceso total-o-nulo, ya que en cualquier instante el impulso de voltaje aparece o no aparece y no tiene objeto una respuesta diferencial o graduada. Una situación análoga sería una en la que nosotros encendemos o apagamos una luz eléctrica sin posibilidad de un interruptor de atenuación que nos permitiera ajustar la intensidad. Es más, la célula no disparará cada vez que reciba algún estímulo sobre sus dendritas, ya que es preciso que la cantidad total de actividad de entrada sea mayor que un nivel residual mínimo (nivel de umbral) antes de que tenga lugar la activación de la célula.

La otra característica interesante del mecanismo de transferencia de información es el hecho de que las conexiones (llamadas *sinapsis*) entre neuronas (generalmente la unión entre el axon de una neurona y la dendrita de la siguiente) se pueden activar de dos formas diferentes. Una conexión llamada *excitativa* tenderá a incrementar la probabilidad de que la célula receptora dispare (es decir, produzca un componente aditivo en la acumulación del estímulo total de entrada) mientras que una conexión *inhibidora* tiene el efecto de disminuir la probabilidad de que la célula dispare (es decir, produzca un componente detractor en la acumulación para alcanzar el umbral de disparo).

UN PROTOTIPO DE NEURONA

A pesar de que algún lector que tenga un conocimiento real de la fisiología de una célula se pueda sentir ligeramente ofendido por la simplificación del proceso que presentaremos más adelante, podríamos argumentar que la descripción ofrecida comprende todas las características *funcionales* importantes del comportamiento de la célula y facilita la información que necesitamos para observar sus propiedades de cálculo. Aceptado esto, podemos representar las características básicas de la célula a este nivel de cálculo de la forma ilustrada en la figura 11.2. Aquí hemos llamado E (expresado con el signo →) a las entradas excitativas de las células e I a las entradas inhibidoras (expresadas con el signo —o). Si suponemos que una línea que está activa (es decir, que ha recibido o generado un impulso de disparo) puede ser considerada con el

estado "1", es decir, E, I ó f = 1 y, de forma análoga, considerar una línea de señal inactiva como estado "0" (recuerde que estamos representado la célula como un dispositivo, disparando o no, esencialmente binario) podemos resumir la operativa de la célula como sigue:

$$f = 1$$

si:

$$(E_1 + E_2 + \dots E_m) - (I_1 + I_2 + \dots I_n) \geqslant T$$

pero:

$f = 0$ de otra forma

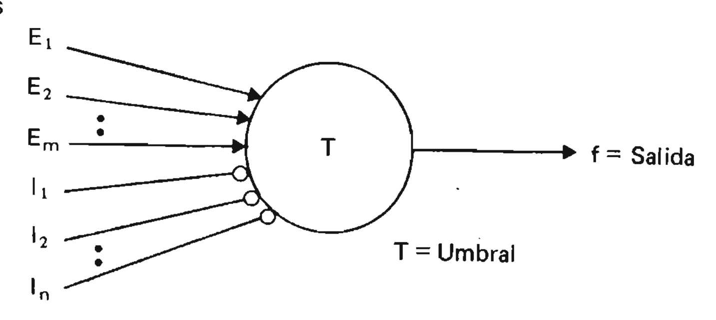


Fig. 11.2.—Modelo de célula simple.

Hemos descrito la célula en términos de sus parámetros funcionales. Ahora podemos comenzar a ver cómo estas células son capaces de ejecutar tareas de proceso que pueden comprenderse como el cálculo de una gama de funciones diferentes.

En la figura 11.3 se muestran algunas posibles configuraciones de células simples. Examine la figura 11.3 (a) a modo de ejemplo. Aquí podemos ver que en el supuesto de que una o ambas de sus dos fibras de entrada estén estimuladas (es decir, reciban impulsos de disparo) la

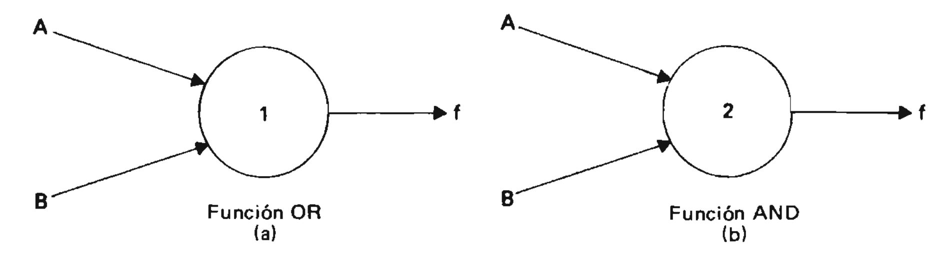


Fig. 11.3.—Algunas funciones típicas de una célula.

célula disparará por sí misma, pero cuando ninguna de las fibras esté activada, la célula permanecerá inactiva. Por otro lado, para la configuración de la figura 11.3 (b), esta célula solamente disparará si sus dos entradas se estimulan simultáneamente. No le llevará mucho tiempo ver que estas dos células están calculando respectivamente las funciones que se presentaron en el capítulo 5 como funciones OR y AND.

Por lo tanto hemos llegado a la conclusión de que si tomamos una célula de cálculo modelada con las propiedades funcionales de la neurona biológica podemos describir su comportamiento utilizando ideas, términos y notaciones que ya conocemos del lenguaje normal de cálculo. Sin embargo, podríamos ampliar estas ideas considerando algunos ejemplos de pequeñas "redes" de células interconectadas. Considere, como ejemplo, la red mostrada en la figura 11.4 (a). Un análisis de esta red revelará que señalizará cuando una o dos de sus entradas A, B y C

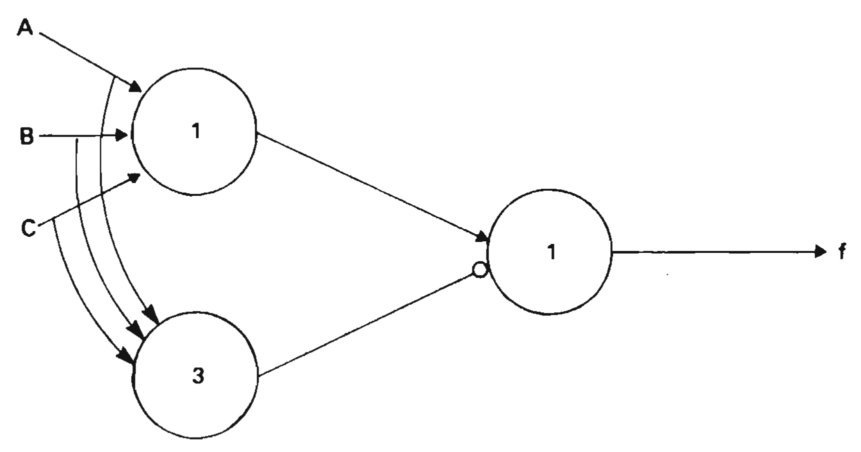


Fig. 11.4 (a).

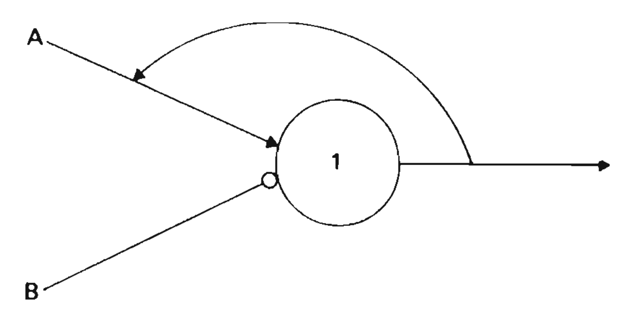


Fig. 11.4 (b).

estén activas, pero suprimirá su salida (la dejará inactivada) si las tres entradas se activan al tiempo. La configuración de la figura 11.4 (b) se incluye como ejercicio para que pruebe Vd. solo. ¿Qué hace esta red?

UN MODELO DE NEURONA MAS FLEXIBLE

El tipo de neurona que hemos presentado y tratado hasta ahora es muy limitado y no muy flexible, siendo más usual imitar el proceso de las neuronas utilizando la versión ligeramente modificada que se ilustra en la figura 11.5. Existen dos diferencias principales entre esta versión del tipo de neurona y la anterior. La primera es que aquí hemos introducido un *factor de ponderación* (w) que modifica el efecto de una entrada activada sobre la neurona receptora, de forma que cuanto más grande es el valor de una ponderación en particular, más importancia cobra la entrada correspondiente, y cuanto más pequeño es el valor, menor importancia cobra esa entrada en particular. El segundo cambio es que ya no necesitamos identificar las entradas especificadas como excitativas o inhibidoras en su naturaleza, pudiendo seleccionar cualquiera de las formas, disponiendo simplemente que una entrada tenga una ponderación asociada positiva para un efecto excitativo, o negativo para un efecto inhibidor.

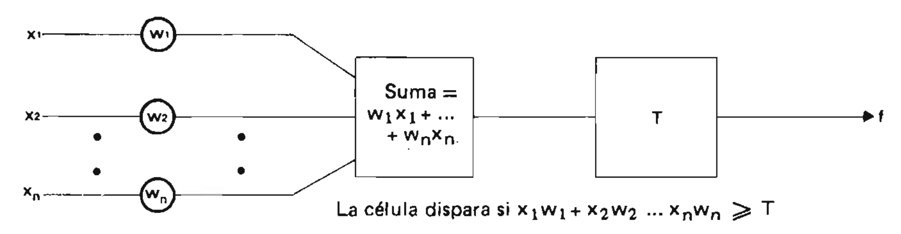


Fig. 11.5.—Un modelo alternativo de célula.

Para esta versión del tipo de neurona, podemos utilizar la siguiente expresión para describir su operación:

$$f = 1$$

si:

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \geqslant T$$

y:

$f = 0$ en caso contrario.

Esta es una versión del modelo más clara que la que tomamos en un principio.

Por ejemplo, podemos realizar una red *aprendizaje* (si quiere expresarlo Vd. así), cambiando los valores de las ponderaciones y por supues-

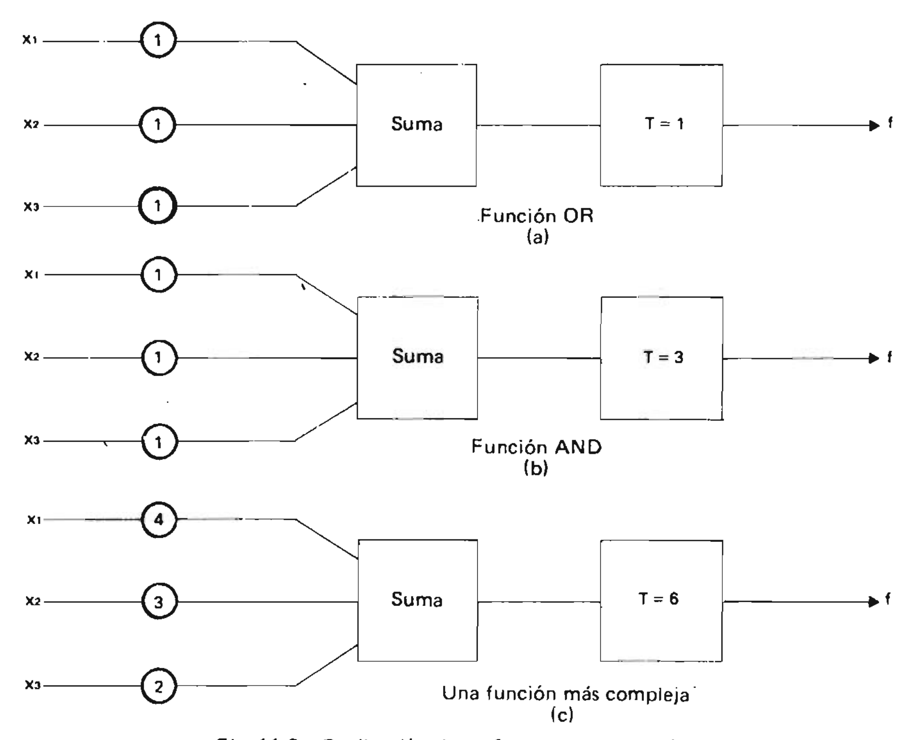


Fig. 11.6.—Realización de las funciones de una célula.

to, es posible obtener funciones de cálculo diferentes diseñando configuraciones de células diferentes. Por ejemplo, en las configuraciones que se muestran en la figura 11.6 (a) y (b) aparecen de nuevo las funciones OR y AND respectivamente, mientras que la figura 11.6 (c) muestra un sistema que ... bueno, ¿qué es lo que hace?

Sin embargo, existe otra forma de examinar este modelo que puede ser mejor comprendida si consideramos un ejemplo específico. Por sencillez, escogeremos una célula que tenga exactamente dos fibras de entrada, como se muestra en la figura 11.7. Primeramente podemos examinar de nuevo las expresiones que determinan si la célula disparará para un patrón en particular de estímulo de entrada. En este caso, la ecuación general presentada anteriormente se reduce a la siguiente expresión específica:

$$f = 1$$

si:

$$w_1 x_1 + x_2 x_2 \geqslant T$$

y:

$f = 0$ en caso contrario.

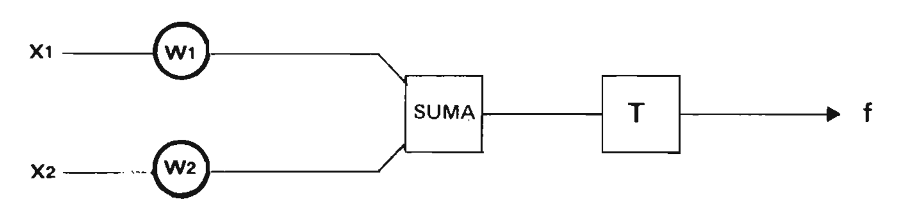


Fig. 11.7.—Ejemplo de una célula generalizada de dos entradas.

En segundo lugar, podemos dibujar una representación del "espacio de entrada" con una especie de gráfico con x_1 y x_2 como ejes (figura 11.8). Vemos que de esta forma cualquier patrón de estímulo en particular (es decir, ambas entradas activas, x_1 activa con x_2 inactiva, etc.) correspondería a un punto específico en esta representación bidimensional. En realidad, en este caso, debido a la naturaleza binaria asumida de la actividad de la neurona para estos puntos, solamente puede existir un equivalente con los puntos $(x_1, x_2) = (0,0)$ ó $(0,1)$ ó $(1,0)$ ó $(1,1)$.

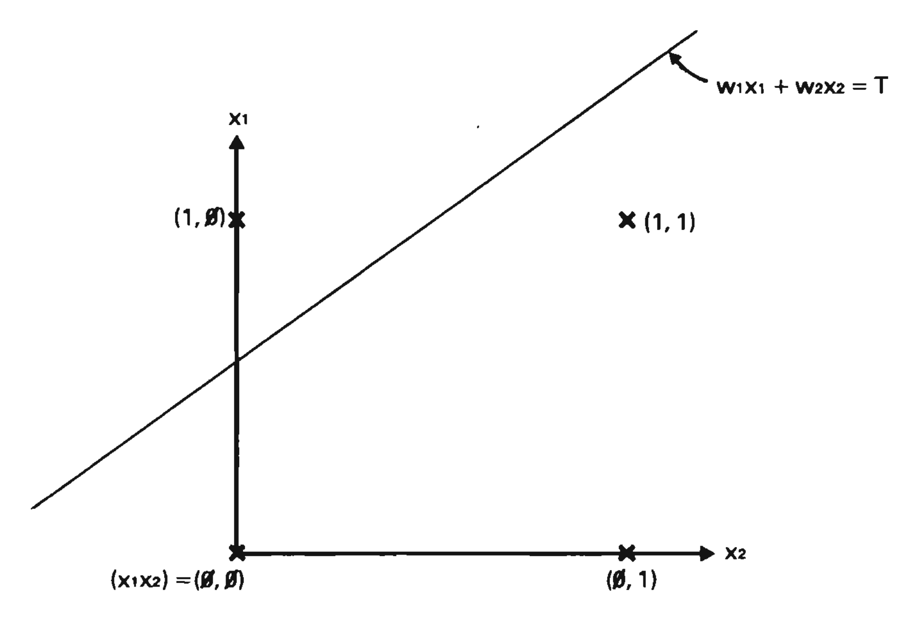


Fig. 11.8.—Espacio de entrada con función discriminante.

Pero si ahora examinamos nuestra expresión anterior en términos de las condiciones requeridas para definir la línea que divide los estados de disparo y no disparo de la célula, vemos que ésta corresponde a la expresión escrita:

$$w_1 x_1 + w_2 x_2 = T$$

Así pues, podemos dibujar esto sobre los ejes que definen nuestro espacio de entrada como antes y ver que la expresión da lugar a una línea recta que divide el espacio en dos regiones. Ahora, un punto dado en el grafo (correspondiente a un patrón particular de estímulo sobre las entradas) generará un estado de disparo/no disparo de la célula con arreglo al lado de la línea donde cae el punto. Lo más importante de todo es que la posición de la línea viene determinada por las ponderaciones w_1 y w_2 (también, por supuesto, por el valor de T, pero si queremos, podemos considerarlo fijo para todas las células).

Verá ahora porqué hemos desarrollado el argumento de esta forma. Para el descubrimiento que hemos realizado (examine de nuevo la última ecuación que escribimos) al modelar la célula básica relacionada con el cálculo biológico, volvemos a una descripción de sus propiedades de

proceso que corresponde exactamente al modo en que formulábamos en el capítulo 3 una aproximación al reconocimiento de patrones sencillos. La única diferencia consiste en que hemos restringido nuestros descriptores de patrones a valores binarios. La otra característica importante del tipo de descripción que tenemos aquí es, por supuesto, que la idea de aprendizaje (es decir, la adaptación de un patrón de comportamiento o el cambio de una función de cálculo como aparecería aquí) corresponde en principio al proceso del cambio de la función de las células individuales. Esto, a su vez, corresponde en nuestro modelo al cambio de los valores dados a las ponderaciones de entrada (asumiendo que fijamos el umbral de la celda), por lo cual obtenemos un mecanismo con el cual modelar la adaptación en una red de células.

Esta aproximación al modelado de las células de las neuronas es instructiva por otra razón, ya que provoca ciertas preguntas interesantes e importantes sobre el tipo de cálculo que puede darse en tales redes. Por ejemplo, en el supuesto anterior yo no pienso que ni el lector matemáticamente más impuesto sea capaz de encontrar un conjunto de valores para w_1, y w_2 tales que la célula dispare cuando una de las fibras de entrada solamente (no ambas) sea activada, pero que permanezca inactiva cuando ambos canales de entrada portan la misma señal (ambas activas o ambas inactivas a la vez). Si Vd. piensa que puede resolver este problema, probablemente se hará famoso. Sin embargo, es más importante, que se pregunte a sí mismo ¿por qué surge esta dificultad? A modo de ejercicio, trate de descubrir cómo se podría llevar a cabo el cálculo descrito, pero de otra forma alternativa, sin cambiar la descripción de las células con las que hemos trabajado.

En este capítulo he tratado de mostrar cómo es posible modelar, al menos a un nivel funcional básico, la operativa de la pieza fundamental de construcción de los sistemas biológicos del cálculo. Aunque todos estamos de acuerdo en que, por más que esforcemos la imaginación, todo esto no produce ninguna revelación repentina sobre lo que realmente es la "inteligencia", ni tampoco descubre el modo más obvio o avanzado para el diseño y construcción de máquinas inteligentes, lo que ciertamente nos aporta es la penetración, el conocimiento de los procesos que subyacen en muchos de los aspectos específicos del comportamiento que generalmente consideramos como indicativos de inteligencia. Más importante es quizás, haber demostrado que podemos describir (aunque no explicar necesariamente en términos del mecanismo real) muchos aspectos del proceso de las neuronas, utilizando el tipo de técnicas descriptivas que son muy familiares para los científicos e ingenieros de ordenadores. Finalmente, utilizando esta aproximación, hemos visto cómo, a pesar de que comenzamos desde un punto de vista muy distinto, hemos vuelto a las exposiciones presentadas en el capítulo 3, demos-

trando que algunas de las funciones importantes de las redes semejantes a las de las neuronas dan origen, de forma natural, a las propiedades de proceso fundamentales en actividades inteligentes básicas, tales como el reconocimiento y la clasificación de los patrones de estímulo.

En término de arquitectura, el tipo de sistema de cálculo descrito en este capítulo está mucho más próximo al procesador "inteligente" bien conocido por todos, que a un ordenador digital convencional. Quizás por esta razón es importante que no caigamos fácilmente en la forma de pensar que asume que el ordenador digital en su forma actual facilitará, en última instancia, las respuestas a todas las preguntas que tratamos de buscar sobre la inteligencia y el diseño de una generación de máquinas inteligentes.

12
Breve examen de la anatomía del ordenador

Si volviera a repasar todas las anteriores páginas de este libro quizás le llame la atención un hecho destacable. Supongo que estoy exagerando realmente, porque a menos que esté errado, el hecho al que me refiero es muy probable que no haya pasado desapercibido para la mayoría de los lectores y quizá para la mayor parte de la comunidad de usuarios de ordenadores.

Pero, ¿cuál es este hecho? Simplemente que en todo lo anteriormente expuesto hemos asumido que el ordenador digital convencional es el dispositivo de cálculo más apropiado (probablemente el *único*) para la apliacación a tareas de inteligencia artificial. El hecho de haber suscitado este tema, probablemente le hará desconfiar, por supuesto; en particular puede que ahora esté preguntándose cosas como:

¿Significa que un IBM PC sería mejor que un Spectrum?

¿Existe otro tipo de ordenador que yo podría haber utilizado?

¿Qué razón hay para pensar en otras alternativas cuando a lo largo de todo el libro se ha tratado de explicar cómo utilizar el ordenador convencional para la inteligencia artificial?

¿Tiene este hombre la suficiente experiencia práctica?

... y quizás otras más.

Permítame que le diga antes de nada que estaría Vd. en lo cierto al suponer que el ordenador digital convencional ha sido, todavía es, y probablemente continúe siendo utilizado para el tipo de tareas que hemos discutido hasta ahora. Sin embargo, parece una buena idea al finalizar este libro presentarle otras posibilidades. Comencemos por examinar algunas conclusiones y luego, por medio de un ejemplo anteriormente mencionado, tratar de sugerir algo más específico.

REVISION DEL PROCESO DE PATRON

Vamos a ilustrar nuestra discusión considerando el tipo de problema con el que nos podemos encontrar al manipular patrones que representan datos visuales. Por ejemplo, la figura 12.1 (a) muestra una representación del carácter alfabético **O** en dos dimensiones, pero como puede ver, el proceso de extracción de este dato de su fuente original (un sobre de correo u otro documento, análogo, quizás) ha sido afectado más bien negativamente por el "ruido" o por cambios aleatorios en el medio ambiente.

Sería interesante que, antes de intentar procesar este carácter, pudiéramos limpiar un poco su apariencia. Concretamente, algo útil que podríamos hacer sería intentar eliminar del cuadro cualquier punto negro aislado (o puntos "1" en lugar de puntos "0" en nuestra representación binaria normal). En un desarrollo normal de acontecimientos, podríamos intentar llevarlo a cabo, desarrollando un programa adecuado para un ordenador digital. Brevemente, lo que haríamos sería examinar cada uno de los ocho vecinos inmediatos (Norte, Noreste, Este, Sureste, Sur, Suroeste, Oeste, Noroeste) de un punto negro arbitrario (digamos el punto P) del patrón y calcular si alguno de estos vecinos es un punto negro. Cualquier vecino negro bastará para salvar de la extinción a ese punto P del cuadro en particular, ya que asumiríamos que una conexión a otro punto negro indica que P es parte de la información importante del cuadro, no ruido de ambiente, pero si P tiene un juego completo de ocho vecinos todos ellos blancos, tomamos P como punto de ruido aislado y lo eliminamos (cambiamos a 0 su valor). Desgraciadamente, así tendríamos que repetir este procedimiento tedioso para cada punto negro del cuadro, e incluso aunque un punto fuera blanco tendríamos que efectuar una pequeña comprobación para cerciorarnos de ello.

En una situación práctica, donde probablemente vamos a tratar un patrón compuesto quizás por un mínimo de 64 × 64 puntos (y a veces muchos más) para llevar a cabo incluso esta simple tarea, tendríamos que examinar al menos 4096 elementos del cuadro, de la representación, y realizar un cálculo bastante extenso (incluyendo la búsqueda y comprobación del valor de ocho vecinos) sobre todos los que sean negros.

Por supuesto, se percatará de que este procedimiento es muy semejante al tipo de operaciones previas al proceso descritas anteriormente en el capítulo 10 y además implica el tipo de rutina de "presentación en ventanas" desarrollado allí para adelgazamiento de patrones. También recordará que dichas operaciones no eran demasiado rápidas en su eje-

cución. Recuerde también, que en un ambiente típico de proceso de imágenes probablemente vamos a ejecutar esta operación además de otras muchas operaciones y transformaciones potencialmente sobre un gran número de patrones (es decir, la inspección automática de componentes en una línea de producción, imágenes médicas, etc.), y verá porqué, en los últimos años, los pensamientos se han centrado en las cuestiones fundamentales sobre si puede ser provechoso considerar estructuras alternativas de procesos. Alguien puede argumentar que los informáticos habían estado tanto tiempo esperando resultados que acabaron contemplando otras posibilidades simplemente para pasar el rato.

Y por supuesto, es inevitable que hayan evolucionado otras soluciones alternativas. En particular, ha surgido un tipo de estructura que consideramos de interés por dos razones distintas. En primer lugar, porque es muy eficaz para reducir la magnitud del problema que acabamos de describir al obtener mayores velocidades de proceso y, en segundo lugar, porque esta estructura comporta una más estrecha semejanza con las estructuras encontradas en el cerebro que con las de un ordenador convencional. Esto enlaza muy bien con lo que dijimos en el capítulo anterior y comoquiera que comenzamos nuestra discusión en el capítulo 1 estableciendo que el cerebro es la unidad de medida de la inteligencia, finalizaremos este libro de forma razonablemente satisfactoria.

UNA ESTRUCTURA DE CALCULO ALTERNATIVA

Examinemos pues un sistema de cálculo que entra dentro de esta categoría más bien especializada. Si tomamos como modelo, como patrón, el cerebro (si bien debo subrayar que el sistema que vamos a describir no pretende ser una réplica del cerebro como tal) recordemos que con-



tiene una célula básica de cálculo llamada neurona, que puede calcular una función relativamente sencilla, pero que trabajando simultáneamente con otras muchas células semejantes interconectadas produce unos resultados realmente impresionantes. Ahora conecte esta información con la observación que hemos considerado en muchos de los procesos prácticos de patrones. Un patrón ha consistido en un array bastante regular de puntos y en una operación típica de proceso repetimos en todos los puntos del array un cálculo muy semejante. Por ejemplo, si aplicamos al patrón de la figura 12.1 (a) el algoritmo anterior de la "eliminación del punto de ruido aislado", llevamos a cabo nuestra operación definida, para cada punto del array para producir el nuevo patrón de la figura 12.1 (b).

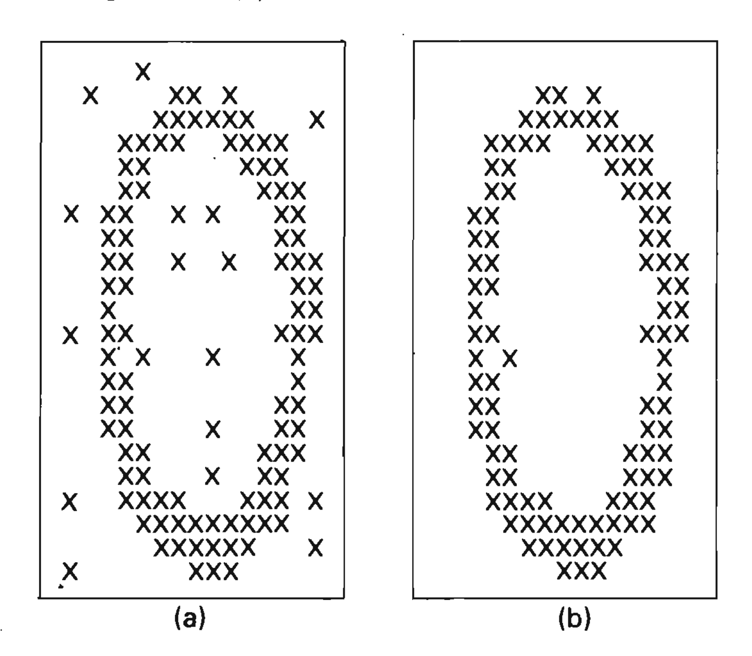


Fig. 12.1.—Patrones representando datos visuales.

Agrupe todas estas ideas y será casi obvio sugerir que quizás deberíamos tomar una célula sencilla individual responsable de un número restringido de operaciones asignando dicha célula como responsable para *cada* punto del array. Si bien tendríamos que invertir más dinero en equipo de procesador (duplicando el número de elementos de cálculo requerido) podríamos hacer que cada procesador fuera mucho más sim-

ple que un microprocesador de propósito general y, como todas las células pueden ser idénticas, su fabricación no sería demasiado cara. ¡Piense en lo rápido que podría llevarse a cabo una operación! Esto se debe a que todas las células pueden trabajar simultáneamente y, por tanto, cada punto del array podría procesarse al mismo tiempo. Si escogemos para nuestra célula un diseño y algoritmos de proceso apropiados, podemos disponer que, en cada instante, cada célula esté ejecutando la misma operación. Esto puede comportar ventajas enormes a la hora de programar nuestro nuevo sistema de proceso.

Así pues, nuestro nuevo procesador se asemejaría a lo expuesto en la ilustración de la figura 12.2, donde tenemos un *array* (en este caso un array cuadrado, aunque son posibles otras configuraciones) donde cada elemento es una célula simple de proceso y a cada una corresponde exactamente un punto del patrón a procesar. Esta estructura se conoce a veces por el nombre de *array iterativo* y adopta el principio de *proceso paralelo* (donde cada célula calcula simultáneamente, en paralelo, una función idéntica) en lugar del proceso en serie del ordenador convencional. Por supuesto, a pesar de haber restringido en

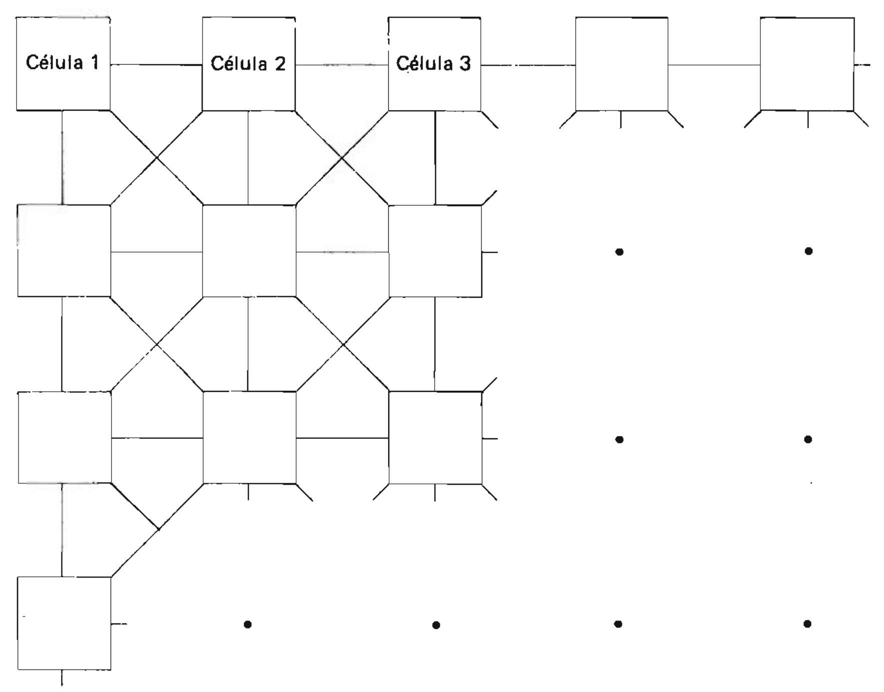


Fig. 12.2.—Un array.

cierta medida la disposición de las células en el array (un trazado regular, correspondencia estricta entre un punto del patrón y una célula de proceso, todas las células calculan una función idéntica, etc.), en principio, tiene muchas características que identificamos al considerar la estructura del array iterativo lo que lo hace especialmente interesante para nosotros.

DISEÑO DE LAS CELULAS BASICAS DE PROCESO

Ahora debemos volver a la cuestión del diseño de las células de proceso individuales, ya que esto es materia de crucial importancia. Se puede asegurar que podrían adoptarse muchos tipos diferentes de estructuras de procesador para las células individuales, pero a fin de ilustrar algunas características generalmente aplicables, supongamos que dentro de cada célula tenemos los componentes mostrados en la figura 12.3. Los componentes principales serían:

1. La célula tendrá que recibir información relativa al valor actual del punto con el que está asociado. Esta información pasa a la célula por la entrada denominada I.

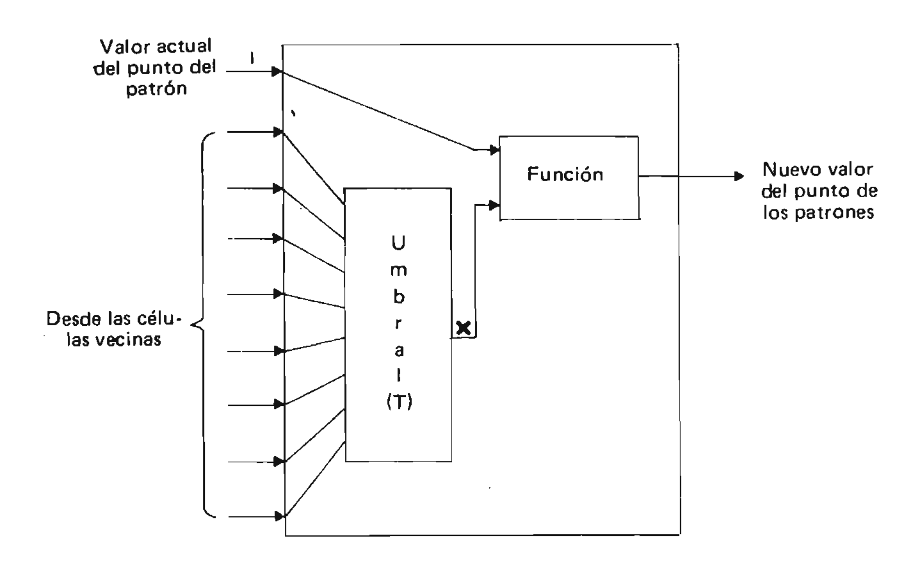


Fig. 12.3.—Diseño de una célula básica de proceso.

2. Generalmente, una operación sobre un punto del cuadro necesitará conocer no sólo el valor de ese punto en particular, sino también los valores de los puntos vecinos inmediatos, que normalmente serán ocho. Para simplificar el cálculo de la célula, vamos a suponer que tenemos un dispositivo (UMBRAL) que detectará, haciendo X = 1, cuando T o más vecinos están almacenando un valor "1" (es decir, conectados a puntos negros). Por supuesto, debemos poder seleccionar un valor deseado para T.
3. El bloque rotulado "FUNCION" es un dispositivo capaz de detectar cualquiera de las dos entradas X e I a fin de calcular el siguiente valor del elemento patrón asociado a una célula en particular. Desearíamos poder seleccionar una función apropiada en cualquier momento, por lo que necesitamos algunas entradas de selección. Como verá, este dispositivo es muy semejante a los que nos referimos en el capítulo 5.

Por tanto, la "programación" de este array se corresponde con la selección de umbrales y funciones apropiados para transformar en cada instante el conjunto de puntos de patrones y, de forma exactamente igual que sucedería con programas de ordenador convencionales, podríamos necesitar varios pasos para llevar a cabo algún proceso requerido.

Si Vd. está pensando que esta configuración de célula es arbitraria, volveremos ahora al problema de la "eliminación de ruido" y mostraremos cómo este array puede resolver la tarea.

En realidad, va a ser algo muy sencllo. En primer lugar, establecemos el umbral T a "1" para asegurar que X se convierte en "1" si cualquiera de los ocho vecinos de una célula en particular están asociados con un punto de patrón negro. Solamente X será igual a "0" en el caso de que todos los vecinos estén asociados con puntos de patrón blancos. Luego, seleccionamos nuestra vieja amiga, la función "AND" de la casilla FUNCION (ya nos hemos encontrado muchas veces con esta función).

Tomemos una célula específica y veamos lo que sucede al activarla para que realice su cálculo (esto sucedería generalmente en la práctica aplicando una señal de control al array). Si el punto del patrón asociado con la célula es negro (haciendo I = 1) y al menos uno de los puntos vecinos es negro (haciendo X = 1), la casilla de funciones, ahora conteniendo la función AND que hemos seleccionado, sacará un "1" (ya que I y X son ambas "1") asegurando así que el punto del patrón asociado con la célula sigue siendo un punto negro.

Si el punto del patrón es un punto blanco, entonces I = 0 y la salida de la función AND será "0" con independencia del valor de X, dejando

blanco el punto. Finalmente, si el punto del patrón es negro (I = 1) y ninguno de los puntos vecinos es negro (haciendo X = 0), la salida de la casilla FUNCION será "0" (blanco) y cambiará el punto del patrón de negro a blanco.

Recuerde que este procedimiento se ejecutará en cada punto del array del patrón simultáneamente. Este comportamiento es precisamente lo que se necesita para llevar a cabo el algoritmo de eliminación de ruidos que hemos descrito, pero con la ejecución de una única instrucción (emitida simultáneamente para todas las células) en lugar de la secuencia de operaciones, con un gran consumo de tiempo, que se requerían cuando intentamos implantar el mismo algoritmo sobre nuestro ordenador convencional de serie.

Por supuesto que éste es sólo un tipo de operación, pero es fácil ver cómo se podrían construir otras instrucciones, particularmente si ampliamos un poco las posibilidades de cada célula.

Este capítulo ha demostrado como, con un poco de imaginación, podríamos presentar una nueva dimensión para la solución de algunos problemas típicos del proceso de patrones y sugerir una aproximación más amplia para la construcción de sistemas de proceso "inteligente". Desde nuestro punto de arranque en la discusión sobre la naturaleza de la inteligencia en relación con la capacidad humana, hemos pasado por ejemplos de problemas típicos sobre inteligencia artificial, y ahora hemos llegado a un punto en el que hemos vuelto a considerar el cerebro humano para obtener algunas ideas acerca de la solución de algunos de esos problemas. Quizás una de las peculiaridades más excitantes de este área de trabajo reside, en última instancia, en el hecho de que aunque los problemas que estamos abordando son fundamentales para las actividades normales de los seres humanos, son con frecuencia tremendamente difíciles de definir con claridad y de forma conveniente para la solución con ordenador. Por otra parte, cada uno de los problemas que hemos abordado entra dentro de los límites de una serie de disciplinas diferentes y esto en sí mismo es un reto en el campo de la inteligencia artificial.

Confío que con este examen a modo de introducción Vd., el lector, experimente esta sensación de reto y desee ejercitar su ingenio para explorar y ampliar estas ideas. Después de todo, eso es seguramente lo que hace ser tan especiales a los seres humanos; tienen la posibilidad de ejercitar la inteligencia en el estudio de la propia inteligencia.

Apéndice 1: PATREC-Sistema de Reconocimiento de Patrones de Red de Aprendizaje

Programa principal

```
  1 CLEAR 31999
  2 LET limpia=2600: LET editar=2800: LET copiar=3000: LET lanzar=3200:
    LET retin=2400
 10 LET ponc=3400: LET tomabits=2200: LET sacabit=2000: LET and=32012:
    LET inici=1000: LET enseñar=1200: LET recon=1400:
    LET bina=1600:
    LET pad=1700: LET ponbit=1800: LET or=32002
 20 DIM r$(12,8): DIM c(96,2): DIM n$(32)
 30 FOR p=32002 TO 32021
 35 READ byte
 40 POKE p,byte
 45 NEXT p
 50 DATA 6,0,33,0,125,126,35,182,79,201,6,0,33,0,125,126,35,166,79,201
 90 GO SUB ponc: GO SUB inici
100 GO SUB enseñar
110 LET num=CODE n$(1): GO SUB bina: PRINT b$,
120 LET num=CODE n$(2): GO SUB bina: PRINT b$
130 INPUT "mode";m$
140 IF m$="t" THEN  GO TO 100
150 GO SUB recon
160 GO TO 130
```

inici (1000)

Función: Inicializa el array n$ a nulos.

```
1000 FOR p=1 TO 32
1010 LET n$(p)=CHR$ 0
1020 NEXT p
1030 RETURN
```

enseñar (1200)

Función: Acepta una paterna de la retina de visualización y un señalizador de clasificación. Modifica n$ adecuadamente.

```
1200 GO SUB retin
1210 INPUT "clasificacion(0=no-5,1=5)";señal
1220 IF NOT señal THEN  RETURN
1230 FOR p=1 TO 32
1240 GO SUB tomabits
1250 GO SUB ponbit
1260 NEXT p
1270 RETURN
```

recon (1400)

Función: Acepta una paterna de la retina de visualización y determina si es un miembro del tipo de reconocimiento.

```
1400 CLS : LET tot=0
1410 PRINT AT 20,0;"Entre figura a reconocer"
1420 GO SUB retin
1430 FOR p=1 TO 32
1440 GO SUB tomabits
1450 GO SUB sacabit
1460 IF bit>0 THEN  LET tot=tot+1
1470 NEXT p
1480 IF tot=32 THEN  PRINT "5"
1490 RETURN
```

bina (1600)

Función: Acepta un entero positivo en num y devuelve su equivalente binario como una cadena en b$; b$ se garantiza que sea al menos de 6 caracteres de longitud.

```
1600 LET b$=""
1610 IF NOT num THEN  GO SUB pad: RETURN
1620 LET int=INT (num/2)
1630 LET rem=num-2*int
1640 LET b$=STR$ rem+b$
1650 LET num=int
1660 GO TO 1610
1700 IF LEN b$>=6 THEN  RETURN
1710 LET b$="0"+b$
1720 GO TO 1700
```

ponbit (1800)

Función: Acepta en s$ una cadena binaria y hace un OR al byte de orden p de n$ con un "1" en el bit de orden s$.

```
1800 POKE 32000,2^VAL ("BIN "+s$)
1810 POKE 32001,CODE n$(p)
1820 LET n$(p)=CHR$ (USR or)
1830 RETURN
```

sacabit (2000)

Función: Acepta en s$ una cadena binaria y hace un AND al byte de orden p con un "1" en el bit de orden s$. Devuelve el resultado en bit.

```
2000 POKE 32000,2^VAL ("BIN "+s$)
2010 POKE 32001,CODE n$(p)
2020 LET bit=USR and
2030 RETURN
```

tomabits (2200)

Función: Forma una dirección binaria en s$ para el elemento de orden p de n$ utilizando apuntadores en el array C.

```
2200 LET s$=""
2210 FOR b=1 TO 3
2220 LET fila=3*(p-1)+b
2230 LET s$=r$(c(fila,1),c(fila,2))+s$
2240 NEXT b
2250 RETURN
```

retin (2400)

Función: Permite al usuario crear o editar una paterna sobre la retina de visualización.

```
2400 INPUT "Nuevo,Editar,Salir";o$
2410 IF o$<>"n" AND o$<>"e" AND o$<>"s" THEN  GO TO 2400
2420 IF o$="s" THEN  RETURN
2430 IF o$="n" THEN  GO SUB limpia: GO SUB editar: GO TO 2400
2440 GO SUB editar
2450 GO TO 2400
```

limpia (2600)

Función: Limpia la retina r$ con ceros ASCII.

```
2600 FOR y=1 TO 12
2610 LET r$(y)="00000000"
2620 NEXT y
2630 RETURN
```

editar (2800)

Función: Permite al usuario editar la visualización actual de la retina utilizando las teclas de cursor para mover un cursor "#" y la

tecla EDIT para alterar un pixel entre activado y desactivado. (Todas las teclas se utilizan sin SHIFT).

```
2800 FOR y=1 TO 12 STEP 2
2810 FOR x=1 TO 8 STEP 2
2820 PRINT  PAPER 4;AT y,x;" ";AT y+1,x+1;" "
2830 PRINT  PAPER 5;AT y,x+1;" ";AT y+1,x;" "
2840 NEXT x
2850 NEXT y
2860 GO SUB copiar
2870 LET x=1: LET y=1
2880 PRINT  OVER 1;AT y,x;"^"
2890 LET c$=INKEY$
2900 IF c$="" THEN  GO TO 2890
2905 FOR q=1 TO 20: NEXT q
2910 IF CODE c$=13 THEN  RETURN
2920 IF c$="1" THEN  GO SUB lanzar
2930 IF c$="5" AND x>1 THEN  PRINT  OVER 1;AT y,x;"^": LET x=x-1: GO TO 2880
2940 IF c$="6" AND y<12 THEN  PRINT  OVER 1;AT y,x;"^": LET y=y+1: GO TO 2880
2950 IF c$="7" AND y>1 THEN  PRINT  OVER 1;AT y,x;"^": LET y=y-1: GO TO 2880
2960 IF c$="8" AND x<8 THEN  PRINT  OVER 1;AT y,x;"^": LET x=x+1: GO TO 2880
2970 GO TO 2880
```

copiar (3000)

Función: Copia el estado actual de r$ en la visualización de la retina.

```
3000 FOR y=1 TO 12
3010 FOR x=1 TO 8
3020 IF r$(y,x)="1" THEN  PRINT AT y,x;" "
3030 NEXT x
3040 NEXT y
3050 RETURN
```

lanzar (3200)

Función: Lanza un pixel a r$ y a la pantalla bajo control de edit.

```
3200 LET r$(y,x)=STR$ NOT VAL r$(y,x)
3210 IF r$(y,x)="1" THEN  PRINT AT y,x;" "
```

```
3220 IF r$(y,x)="0" THEN  PRINT AT y,x;" "
3230 RETURN
```

ponc (3400)

Función: Inicializar el array c de los apuntadores de coordenadas en n$.

```
3400 FOR y=1 TO 12
3410 FOR x=1 TO 8
3420 LET c(8*(y-1)+x,1)=y
3430 LET c(8*(y-1)+x,2)=x
3440 NEXT x
3450 NEXT y
3460 RETURN
```

Apéndice 2: LIZ-Implantación Simplificada de Eliza

Programa principal

```
 10 LET checksub=2400:LET triturador=2200: LET select=2000: LET barquillo=1800:
    LET buscapal=1600: LET parada=1000: LET buscaver=1200:
    LET transobj=1400
 20 DIM c$(3,17): DIM a$(20,10): DIM b$(20,10): DIM q$(4,40): DIM t$(10):
    DIM v$(40)
100 PRINT "Hola, ¿qué quieres decirme?"
110 INPUT s$
115 PRINT  INK 4;s$
120 IF s$(LEN s$)="?" THEN  GO SUB parada: GO TO 110
130 GO SUB buscaver
135 IF p=1 THEN  GO SUB barquillo: GO TO 110
140 LET o$=s$(p TO )
150 GO SUB checksub
160 GO SUB transobj
170 GO SUB select
180 LET k$=r$+o$: GO SUB triturador: PRINT l$
190 GO TO 110
```

parada (1000)

Función: Imprime una cadena al azar de q$.

```
1000 LET r=INT (RND*4)+1
1010 PRINT q$(r)
1020 RETURN
```

buscaver (1200)

Función: Acepta una frase o una sentencia en s$ y devuelve en t$ el verbo más a la derecha (si se encuentra alguno), y un apuntador al verbo en p. Si no se encuentra ningún verbo, al volver p = 1.

```
1200 LET ep=LEN s$
1210 GO SUB buscapal
1220 LET t$=w$
1230 FOR v=1 TO 100
1240 IF t$=v$(v) THEN  RETURN
1270 NEXT v
1280 LET ep=p-2
1290 IF p>1 THEN  GO TO 1210
1300 RETURN
```

transobj (1400)

Función: Acepta el objeto de una frase en p$ y vt que es falso o verdadero dependiendo de si es necesaria una transformación de verbo. El objeto transformado se devuelve en o$.

```
1400 LET ep=LEN o$
1405 LET st=3
1406 IF vt THEN  LET st=1
1410 LET s$=o$: LET o$=""
1420 GO SUB buscapal
1430 IF w$="un " OR w$="una " OR w$="alguno " OR w$="algun "
     THEN  LET w$="el ": GO TO 1480
1435 LET t$=w$
1440 FOR a=st TO 4
1450 IF t$=a$(a) THEN  LET w$=b$(a): GO TO 1480
1460 IF t$=b$(a) THEN  LET w$=a$(a): GO TO 1480
1470 NEXT a
1480 LET o$=w$+o$
1490 IF p>1 THEN  LET ep=ep-1: GO TO 1420
1500 RETURN
```

buscapal (1600)

Función: Acepta un apuntador en ep al final de una palabra en s$ y devuelve la palabra anterior en w$ junto con los apuntadores p y ep a su comienzo y final respectivamente.

```
1600 LET p=ep-1
1610 IF p>0 THEN  IF s$(p)<>" " THEN  LETp=p-1: GO TO 1610
1620 LET p=p+1
1630 LET w$=s$(p TO ep)
1640 RETURN
```

barquillo (1800)

Función: Genera una frase en l$ y la imprime cuando buscaver falla en la búsqueda.

```
1800 FOR p=2 TO 10
1810 IF o$(p)=" " THEN  GO TO 1830
1820 NEXT p
1830 LET r=INT (RND*3)+1
1840 LET r$=c$(r)
1850 IF RND>0.5 THEN  LET r$=r$+" sobre porqué Vd. "+o$
1860 LET k$=r$: GO SUB triturador: PRINT l$
1870 RETURN
```

select (2000)

Función: Selecciona al azar una frase de apertura para una respuesta de Liz.

```
2000 LET r=INT (RND*4)+1
2010 LET r$=d$(r)
2020 RETURN
```

triturador (2200)

Función: Acepta una cadena en k$ y elimina espacios extraños, devolviendo el resultado en l$.

```
2200 LET l$=""
2210 FOR p=1 TO LEN k$
2220 IF k$(p)<>" " THEN  LET l$=l$+k$(p): LET espacio=0
2230 IF k$(p)=" " THEN  LET espacio=espacio+1
2240 IF espacio=1 THEN  LET l$=l$+" "
2250 NEXT p
2260 RETURN
```

checksub (2400)

Función: Comprueba el sujeto de la frase en s$. Si es "yo" o "tu" vt se pone a verdadero. En caso contrario vt se pone a falso. Si el sujeto no es "yo", se modifica o$ apropiadamente.

```
2400 LET vt=1
2410 IF s$(p-2)="yo" THEN RETURN
2420 LET ep=p-2: GO SUB buscapal
2430 IF w$="tu" THEN  LET o$="piensas que yo "+o$: RETURN
2440 IF w$="nosotros" OR w$="nosotras" THEN  LET o$="piensais que yo "
    +o$: RETURN
2445 IF w$="ellos " OR w$="ellas "THEN LET o$="piensan que yo "+o$: RETURN
2450 LET o$="pienso que el "+w$+" "+o$: LET vt=0:RETURN
```

LOS ARRAYS "OCULTOS"

A continuación se relacionan los contenidos sugeridos de los arrays que se encuentran deliberadamente ocultos para el usuario.

1. v$: Relación de verbos

soy	eres	es	fue	fueron	serán	tomar	toma	tomó
puedo	oyde	seré	sería	corro	corre	corrí	dejo	deja
hago	hace	hice	llevo	lleva	llevó	tengo	tiene	tuvo
digo	dice	dije	paro	paré	parado	vuelco	vuelca	volcado
cierro	cierra	abro	abre	abierto	limpio	limpia	limpio	tomo
toma	tomo	tomó	miro	mira	mirado	cocino	cocina	cocinar
canto	canta	cantó	repaso	repasa	repasó	lavo	lava	lavado
quiero	quiere	quiso	fijo	fija	fijó	vengo	viene	vino
como	come	comí	bebo	bebe	bebió	leo	lee	hacer
hacemos	hecho	pongo	pone	tiro	tira	tiré	digo	dice
dicho	veo	ve	visto	pienso	piensa	pensado	puedo	puede
gusto	gusta							

2. q$: Frases de parada

Me temo que no puedo responder a éso
No lo sé
No puedo responder a preguntas como ésta
¿Por qué me preguntas esto?

3. a$/b$: Transformaciones de verbos irregulares

soy	eres
fui	fuiste
mi	tu
me	te

4. c$: Frases de "continuar"

Cuéntame más
Por favor continúa
Prosigue

5. d$: Respuestas de frase inicial

Así que tú
Por qué piensas que
Por qué supones que
Desearía saber porqué tú

Apéndice 3: CONOCIMIENTO- Sistema de Representación y Recuperación del Conocimiento

Progama principal

```
10 DIM t$(20): DIM a$(50,2,20): DIM d$(20): DIM m$(2): DIM n$(100,20):
   DIM p(100,2): DIM e$(3,20): DIM q(3)
20 LET menosuno=3200: LET rastreo=3000: LET buscanodo=2800:
   LET conex=2600: LET hijas=2400: LET madre=2200:
   LET crecer=1000: LET buscar=2000
25 LET pff=1
30 INPUT "crecer o buscar(c/b)";s$
40 IF s$="c" THEN  GO SUB crecer
50 IF s$="b" THEN  GO SUB buscar
60 GO TO 30
```

crecer (1000)

Función: Permite que el usuario introduzca tres nodos (madre y dos hijas) que luego se enlazan a un árbol ya existente.

```
1000 INPUT "partes de";e$(1);" son ";e$(2);" y ";e$(3)
1010 FOR i=1 TO 3
1020 LET q(i)=0
1030 FOR r=1 TO 100
1040 IF n$(r)=e$(i) THEN  LET q(i)=r:GO TO 1060
1050 NEXT r
1060 NEXT i
1070 FOR i=1 TO 3
```

```
1080 IF q(i)<>0 THEN  GO TO 1120
1090 LET n$(pff)=e$(i)
1100 LET q(i)=pff
1110 LET pff=pff+1
1120 NEXT i
1130 LET p(q(1),1)=q(2)
1140 LET p(q(1),2)=q(3)
1150 RETURN
```

buscar (2000)

Función: Permite que el usuario examine un árbol para buscar las conexiones entre nodos específicos.

```
2000 INPUT "encontrar  madre,hijas,conexion,salida(m/h/c/s)";q$
2010 IF q$="m" THEN  GO SUB madre: GO TO 2000
2020 IF q$="h" THEN  GO SUB hijas: GO TO 2000
2030 IF q$="c" THEN  GO SUB conex: GO TO 2000
2040 IF q$="s" THEN  RETURN
2050 GO TO 2000
```

madre (2200)

Función: Acepta un nodo en d$ e imprime su madre, si existe.

```
2200 INPUT "encontrar madre de";d$
2210 LET t$=d$: GO SUB buscanodo
2215 IF NOT coinc THEN  PRINT "nodo no encontrado": RETURN
2220 GO SUB menosuno
2230 IF raiz THEN  PRINT "raiz": RETURN
2240 PRINT m$
2250 RETURN
```

hijas (2400)

Función: Acepta un nodo en m$ e imprime sus hijas, si las hay.

```
2400 INPUT "averiguar las hijas de";m$
2410 LET t$=m$: GO SUB buscanodo
```

```
2420 IF NOT coinc THEN  PRINT "nodo no encontrado": RETURN
2430 IF p(i,1)=0 AND p(i,2)=0 THEN  PRINT "esta es una hoja":RETURN
2440 IF p(i,1)>0 THEN  PRINT n$(p(i,1))
2450 IF p(i,2)>0 THEN  PRINT n$(p(i,2))
2460 RETURN
```

conex (2600)

Función: Imprime la distancia (es decir, el número de ramas) entre dos nodos de entrada e indica si están en línea directa.

```
2600 INPUT "primer nodo";t$
2610 GO SUB buscanodo
2620 IF NOT coinc THEN  PRINT "nodo no encontrado": RETURN
2630 LET j=i
2640 INPUT "segundo nodo";t$
2650 GO SUB buscanodo
2660 IF NOT coinc THEN  PRINT "nodo no encontrado": RETURN
2670 LET k=i
2680 LET i=j: LET col=1: GO SUB rastreo
2690 LET ep1=a
2700 LET i=k: LET col=2: GO SUB rastreo
2710 LET ep2=a
2720 IF ep1=0 OR ep2=0 THEN  LET enlinea=1: GO TO 2750
2730 IF a$(ep1,1)=a$(ep2,2) THEN  LET ep1=ep1-1: LET ep2=ep2-1: GO TO 2720
2740 LET enlinea=0
2750 LET distancia=ep1+ep2
2760 PRINT "distancia=";distancia
2770 IF enlinea THEN  PRINT "enlinea"
2780 RETURN
```

buscanodo (2800)

Función: Acepta un nodo en t$ y busca concordancia en el árbol. Si se encuentra una, devuelve coinc = 1 y un apuntador al nodo en i. En caso contrario, coinc = 0.

```
2800 LET coinc=1
2810 FOR i=1 TO 100
```

```
2820 IF n$(i)=t$ THEN  RETURN
2830 NEXT i
2840 LET coinc=0
2850 RETURN
```

rastreo (3000)

Función: Acepta un apuntador i a un nodo y un apuntador col a un camino. Crea un camino desde el nodo apuntado a la raíz y devuelve ésta en a$.

```
3000 LET a$(1,col)=n$(i): LET a=1
3010 GO SUB menosuno
3020 IF raiz THEN  RETURN
3030 LET a=a+1
3040 LET a$(a,col)=m$
3050 GO TO 3010
```

menosuno (3200)

Función: Acepta un apuntador i a un nodo y devuelve el nodo en m$, que apunta a it. Si el nodo no está apuntado, devuelve raíz = 1. En caso contrario raíz = 0. También devuelve i apuntando al nodo anterior.

```
3200 FOR r=1 TO 100
3210 IF p(r,1)=i OR p(r,2)=i THEN  LET m$=n$(r): LET raiz=0: LET i=r: RETURN
3220 NEXT r
3230 LET raiz=1
3240 RETURN
```

Indice alfabético

Libros sobre INFORMATICA publicados por



Obras generales

ANGULO y ZAPATER. Introducción a la informática. 1986
BANKS. Microordenadores. Cómo funcionan. Para qué sirven. 1984
GARCIA SANTESMASES. Cibernética. Aspectos y tendencias actuales. 1980
HUNT. Manual de Informática básica. 1985
SHELLEY. Microelectrónica. 1983
URMAIEV. Calculadores analógicos. Elementos de simulación. 1983

Diccionarios

HART. Diccionario del Basic. (2 Edic.). 1985
NANIA. Diccionario de Informática. Inglés. Francés. Español. Mas de 11000 términos. 1985. Rústica
NANIA. Diccionario de Informática. Misma edic. anterior en Cartoné. 1985
OLIVETTI. Diccionario de Informática. Inglés-Español. (6 Edic.) 1985

Hardware (Equipo físico)

ANGULO. Electrónica digital moderna. (6 Edic.). 1985
ANGULO. Memorias de burbujas magnéticas. Tecnología y sistemas de aplicación. 1982
ANGULO. Microprocesadores. Arquitectura, programación y desarrollo de sistemas. (3 Edic.). 1984
ANGULO. Microprocesadores. Curso sobre aplicaciones en sistemas industriales. (4 Edic.). 1985
ANGULO. Microprocesadores. Diseño práctico de sistemas. (2 Edic.) 1985
ANGULO. Microprocesadores. Fundamentos, diseño y aplicaciones en la industria y en los microcomputadores. (4 Edic.). 1985
ANGULO. Microprocesadores de 16 Bits. El 68000 y el 8086/8088. (2 Edic.). 1985
ANGULO. Prácticas de microelectrónica y microinformática. (2 Edic.). 1985
ASPINALL. El microprocesador y sus aplicaciones. 1984
GARLAND. Diseño de sistemas microprocesadores. (2 Edic.). 1984
HALSALL. Fundamentos de microprocesadores. 1984
LEPAPE. Programación del Z80 con ensamblador. 1985
ROBIN. Interconexión de microprocesadores. (2 Edic.). 1985
RONY. El microprocesador 8080 y sus interfases. 1984

Spectrum

BELLIDO. Cómo programar su Spectrum. (7 Edic.). 1985
BELLIDO. Cómo usar los colores y los gráficos en el Spectrum. (4 Edic.). 1985
BELLIDO. Cómo usar los colores y gráficos del Spectrum. Casete con los programas. 1985
BELLIDO. KIT de gráficos para Spectrum. 1984
BELLIDO. KIT de gráficos (estuche completo con material) 1984
BELLIDO. Spectrum Plus Ultra. La enciclopedia del Spectrum. T. 1. 1985
BELLIDO. Spectrum. Iniciación al código máquina. 1985
BELLIDO. Los trucos del Spectrum. 1986
ELLERSHAW. Las primeras 15 horas con el Spectrum. 1985
ERSKINE. Los mejores programas para el ZX Spectrum. 1985
MARTINEZ VELARDE. El libro del Código máquina del Spectrum (3 Edic.). 1986
WILLIAMS. Programación paso a paso con el Spectrum. 1985

Sinclair QL

FLEETWOOD. Sinclair QL. Guía del usuario. 1985
GALAN PASCUAL. Programación práctica con el Sinclair QL. 1985

IBM-PC

MORRILL. Basic del IBM-PC, IBM PC XT, IBM PC jr. 1985
PLOUIN. IBM-PC. Características. Programación. Manejo. (2 Edic.) 1985

Commodore-64

BATESON. El 64. Más allá del manual. T.1. 1986
BATESON. El 64. Más allá del manual. T.2. 1986
BISHOP. Commodore 64. Programas prácticos. 1985
MONTEIL. Como programar su Commodore 64. T.1. (5 Edic.). 1986
MONTEIL. Como programar su Commodore 64.T.2. (2 Edic.). 1986
RAMON. Manejo y programación del Commodore 64. 1986

Dragón

MURRAY. Programas educativos para el Dragón 32. 1985
BELLIDO. Amaestra tu Dragón. Curso de programación en Basic para el Dragón. 1985
WATT y MANGADA. Basic para niños con el microprocesador Dragón. (2 Edic.). 1986

Atari-520

MARTINEZ VELARDE. Atari-520. Fundamentos. 1986

MSX

PEARCE. MSX. Programación básica. 1985

Lenguajes

BELLIDO y SANCHEZ. Basic para estudiantes. 1985 .....
BELLIDO y SANCHEZ. Basic para Maestros. (2 Edic.). 1985 ....
BELLIDO, SANCHEZ y RODRIGUEZ. Casete con los programas de Basic para Mestros 1985 ....
BURKE. Enseñanza asistida por ordenador. 1986 ....
CUÑAT Pascal para estudiantes. 1986 ....
CHECROUN Basic. Programación de microordenadores. (5 Edic.). 1985 ....
DELANOY. Ficheros en Basic. (2 Edic.). 1985 ....
GALAN PASCUAL. Programación con el lenguaje Cobol. (4 Edic.). 1985 ....
GARCIA MERAYO. Programación en Fortran 77. 1986 ..
LARRECHE. Basic. Introducción a la programación. (5 Edic.). 1985 ....
MARSHALL. Lenguajes de programación para Micros. 1985 ....
MONTEIL. Primeros pasos en Logo. (2 Edic.). 1985 ......
OAKEY Lenguaje Forth para micros. 1985 ....
QUANEAUX. Tratamiento de textos con Basic. 1985 .....
RAMON. 44 Superprogramas en Basic. 1985 ....
ROSSI. Basic. Curso acelerado. (5 Edic.) 1985 ....
SANCHIS y MORALES. Programación con el lenguaje Pascal. (5 Edic.). 1985 ....
WATT y MANGADA Basic para niños. (6 Edic.). 1985 ..
WATT y MANGADA. Basic avanzado para niños. (3 Edic.). 1985 ....

Aplicaciones e informática profesional

ABRAMSON. Teoría de la información y codificación. (5 Edic.) 1981 ....
COHEN Estructura lógica y diseño de programas. 1986
DAX CP/M. Guía de utilización. MP/M, CP/M 86, Wordstar, Basic. 1986 ....
FLORES. Estructuración y proceso de datos. (5 Edic.). 1985 ....
GAUTHIER. Diseño de programas para sistemas. (5 Edic.). 1985 ....
HARTMAN. Manual de los sistemas de información. T.1. (7 Edic.). 1985 ....
HARTMAN. Manual de los sistemas de información. T.2. (7 Edic.). 1985 ....
LEWIS. Estructuras de datos. Programación y aplicaciones. 1985 ....
LUCAS. Sistemas de información. Análisis Diseño. Puesta a punto. 1984 ....
POPKIN. Introducción al proceso de datos. 1986 ....
RODRIGUEZ. Protección de la información. Diseño de criptosistemas informáticos. 1986 ....
SANCHIS. Teoría y construcción de compiladores. 1986
SCHMIDT. Introducción a los ordenadores y al proceso de datos. (5 Edic.). 1985 ....

Inteligencia artificial

ANGULO. Visión artificial por computador. 1986 ....

Robótica

ANGULO. Robótica práctica. Tecnología y aplicaciones. 1985 ....
ANGULO. Curso de Robótica. (2 Edic.). 1985 ....

Varios

ESCUDERO. (Centro de Investigación UAM-IBM). Reconocimiento de patrones. (Fundamentos teóricos, algoritmos y aplicaciones de la moderna técnica denominada «Pattern Recognition». 1977 ....
GOSLING. Códigos para ordenadores y microprocesadores. 1981 ....
PANNELL. El microordenador en la pequeña empresa. 1984 ....
PUJOLLE. Telemática. 1985 ....

INTELIGENCIA ARTIFICIAL CON EL ZX SPECTRUM

¿Qué es la Inteligencia Artificial?

Básicamente, la Inteligencia Artificial hace al ordenador inteligente. Los ordenadores están pensados para:

- el reconocimiento de patrones
- la generación de conversación
- el proceso de imágenes
- la representación del conocimiento.

Con este libro podrá hacerlo.

Esta obra es una introducción a las técnicas y teorías de la Inteligencia Artificial desde los principios básicos, para hacerla asequible a los principiantes.

Todo el desarrollo se ha ceñido a los aspectos de la Inteligencia Artificial que se pueden implantar con facilidad en el ZX Spectrum.

¡Descubra la Inteligencia Artificial y haga pensar a su ZX Spectrum!



Magallanes, 25 - 28015 Madrid

ISBN: 84-283-1488-8